Guiding Large Language Models in Modeling Optimization Problems via Question Partitioning

Xiaotian Pan, **Junhao Fang**, **Feng Wu**[*], **Sijia Zhang**,
Yi-Xiang Hu, **Shaoang Li**, **Xiang-Yang Li**[*]
School of Computer Science and Technology, University of Science and Technology of China
{px259637522, kd1198497461, sxzsj, yixianghu, lishaoa}@mail.ustc.edu.cn,
{wufeng02, xiangyangli}@ustc.edu.cn

Abstract

Optimization problems are ubiquitous across various domains, such as resource scheduling, production planning, and sales management. Traditionally, they are modeled manually, leading to inefficiencies due to difficulties in communication and collaboration between modeling and domain experts. The emergence of Large Language Models (LLMs) has made automated modeling possible. However, real-world applications are often large-scale and have numerous variables and constraints, limiting the applicability of existing methods. To address this, we propose PaMOP, a novel modeling framework based on LLMs, to model optimization problems automatically, given only natural language descriptions. Specifically, we extract and partition the problems using a tree structure, guiding the LLMs to model each set of constraints with self-augmented prompts, thus reducing the demands on the LLM's capabilities of large contents. The mathematical model is then iteratively corrected and validated through our correction procedures. The experiments demonstrate that our method improves performance on the common benchmark dataset NLP4LP, achieving an accuracy of 62.3% and a code executability rate of 86.8% when tested on GPT-4. Additionally, we demonstrate the effectiveness of our PaMOP in handling large real-world problems.

1 Introduction

Optimization problems have extensive and important applications across various fields, such as water resource scheduling [Britz *et al.*, 2013], production and marketing [Sitek and Wikarek, 2018], electric vehicle travel planning [Cuchý *et al.*, 2024], large neighborhood search for bus driver scheduling [Mazzoli *et al.*, 2024], water quality assessment [Burn and McBean, 1985], and chemical product capture and storage [Middleton *et al.*, 2012]. These problems are typically solved using optimization solvers like Gurobi [Gurobi Optimization, 2021], SCIP [Achterberg, 2009], and Xpress [Min *et al.*, 2003], which find the optimal solution by maximizing or minimizing the specified objective function. However, before solving the problem, real-world challenges often arise in the process of modeling. Formulating optimization problems as mathematical expressions from natural language descriptions requires substantial effort, and traditionally, this modeling is done manually by experts. This process demands significant time and effort, as the communication challenges between frontline staff, algorithm developers, and domain experts often hinder the efficiency of accurately "translating" real-world problems into solvable mathematical models.

[*]Corresponding authors.

Large language models (LLMs) have emerged, enabling new ways to optimize problems because of their advanced natural language processing capabilities and strong logical reasoning skills. Currently, LLMs have been applied to optimize the modeling, verification, and validation of optimization problems, building on methods like Chain-of-Thought (CoT) [Wei *et al.*, 2022], Tree of Thoughts [Yao *et al.*, 2024], and Reflexion [Shinn *et al.*, 2024], achieving notable progress [Xiao *et al.*, 2023; AhmadiTeshnizi *et al.*, 2024].

However, guiding LLMs to model optimization problems through prompts still faces several challenges. LLM-generated models often fail to meet expected formatting or content standards, leading to unstable outputs. For instance, LLMs may generate only a textual analysis of the problem or output a mathematical model in markdown instead of a proper code format. This raises our first challenge: **(C1)** *how to get the output that meets the requirements*.

Another significant challenge arises in practical applications, where the scale of the problem limits the LLM's ability to generate accurate models. This results in issues like omitted constraints or confusion between variables and parameters. The length of the input text impacts the LLM's capacity to handle complex logical conditions [Liu *et al.*, 2024]. Current methodologies aimed at augmenting the reasoning capabilities of large models when dealing with extensive texts have proven inadequate in substantially enhancing tasks demanding high precision [Li *et al.*, 2023; Li *et al.*, 2024]. Modeling is among the tasks that are particularly affected by these limitations. As shown in Figure 1, when we directly instruct the LLMs to model a small problem with fewer than 10 parameters and constraints, it demonstrates a significant limitation by omitting some of the constraints. This raises our second challenge: **(C2)** *how to deal* *with large-scale problems*.

Given the precision required to generate an optimization problem model, even minor discrepancies can render the problem unsolvable. While existing correction methods focus on improving the execution success of mathematical models on solvers, they do not address the logical errors within the model, which directly affect the model's accuracy and quality. This leads to our third challenge: **(C3)** *how to improve the accuracy of modeling output*.

All the challenges mentioned above can be attributed to the fact that existing prompts do not fully leverage the modeling capabilities of LLMs. To address these challenges, we propose PaMOP, an automated optimization problem modeling framework (see Figure 2), based on LLMs. We first analyze and partition the problem using a tree structure. Leveraging this partition tree, we guide the LLMs in modeling each constraint set with self-augmented prompts. Finally, aided by insights from the problem analysis, the mathematical model undergoes iterative correction and validation in depth through our correction procedures.

Our key contributions are as follows:

- **PaMOP Framework Design.** We design an automatic optimization problem modeling framework, PaMOP. By providing self-augmented prompts tailored to different problems and constraint categories during modeling, we stabilize and enhance the modeling capabilities of the LLMs, addressing the challenge of incorrect output formats **(C1)**.
- **Problem Partitioning and Tree Structure.** By partitioning problems into independent subproblems using a tree structure, we reduce the LLM's processing burden for long texts and provide auxiliary information for model generation and improvement. This approach also helps mitigate the challenge of problem scale limitations **(C2)**, ensuring more manageable inputs and improving the accuracy of model generation. We develop test cases for large-scale scenarios to demonstrate the efficacy of the proposed method.
- **Experimental Results and Ablation Studies.** Experiments on the NLP4LP dataset demonstrate that PaMOP achieves an accuracy of 62.3% and a code executability rate of 86.8%, both outperforming existing methods. Ablation studies further confirm the importance of using the partition tree in enhancing model performance. These results showcase PaMOP's efficiency and scalability in handling complex optimization problems, particularly in overcoming low-accurate output **(C3)**.

2 Background

An optimization problem is typically characterized by a unique objective function and several constraints. Our goal is to transform the initial problem description into the following mathematical model:

$$\max_{\mathbf{x}} f(\mathbf{x}) \quad \text{s.t.} \quad \forall i \in 1..n, g_i(\mathbf{x}) \leq 0 \tag{1}$$

For a mathematical model of a problem q, it can be expressed as $M = (m_p, m_v, m_o, m_c)$, where m_p represents the constants or parameters in constraints of the problem, m_v represents the decision variables $\mathbf{x}$, m_o denotes the objective function $f(\mathbf{x})$, and m_c represents the constraints of the problem like $g_1(\mathbf{x})$.

Currently, the basic process for using LLMs to model optimization problems is as follows: upon receiving a description of the optimization problem, we first extract key information and structure it in a specific format. Next, prompts are employed to guide the LLMs in constructing the model for the problem. Finally, we iteratively interact with the LLMs to identify and eliminate errors in the generated model.

After completing the modeling, we input the generated model along with the original problem's data file into the solver for the solution. When the output yields an optimal solution that meets the problem's requirements, we consider the modeling to be correct.

3 Method

This section describes the methods of the PaMOP framework (see Figure 2). To construct the partition tree, we use LLMs to extract key information from the problem description. The tree systematically breaks down the problem both mathematically and semantically. Self-augmented prompts guide the LLMs in modeling the leaf nodes, which are merged bottom-up into a complete model. Finally, the root node's information aids in iteratively correcting the model through syntax error correction, code execution, and reverse translation until the model is error-free.

3.1 The Tree of Problem Partitioning

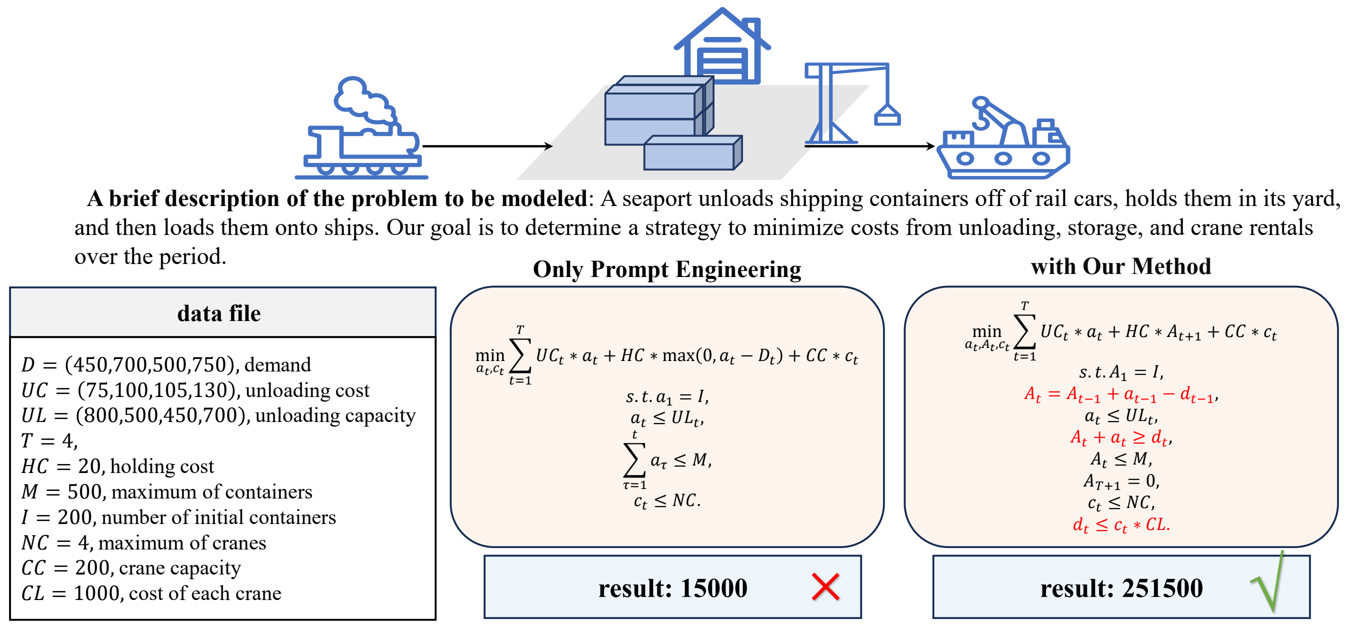
Extracting sub-problems directly from the original problem description is challenging, as it is difficult to obtain independent, model-ready components. Instead, we simulate sub-problems using constraint sets. A tree partitions the problem, creating relatively independent sub-problems, each containing key information needed for modeling. Figure 3 shows the partition result for the example in Figure 1.

The partitioning process occurs layer by layer from the root node downward. The root node represents the entire problem, in which we store all the content of the original problem awaiting partition. At each layer of the partition, a node is split into several child nodes, where the constraints within each child node are highly related. In contrast, the constraints between child nodes are relatively weakly correlated. The leaf nodes formed by the partition represent the sub-problems to be modeled.

The tree structure for partitioning the problem allows us to prevent the LLMs from needing to process complex logical relationships in longer texts. When complex logical processing is necessary (such as in constraint modeling), the LLMs do not need to handle long texts.

3.2 Construction of the Partition Tree

Extracting Structured Representation. Before partitioning, we derive a structured representation of the problem, extracting textual descriptions for the objective functions t_o, constraints t_c, and parameters and variables t_v. To avoid modeling bias from missing the global context, we also generate a concise problem summary g. This extraction process,



A brief description of the problem to be modeled: A seaport unloads shipping containers off of rail cars, holds them in its yard, and then loads them onto ships. Our goal is to determine a strategy to minimize costs from unloading, storage, and crane rentals over the period.

data file

$D = (450{,}700{,}500{,}750)$, demand
$UC = (75{,}100{,}105{,}130)$, unloading cost
$UL = (800{,}500{,}450{,}700)$, unloading capacity
$T = 4$,
$HC = 20$, holding cost
$M = 500$, maximum of containers
$I = 200$, number of initial containers
$NC = 4$, maximum of cranes
$CC = 200$, crane capacity
$CL = 1000$, cost of each crane

Only Prompt Engineering

$$\min_{a_t, c_t} \sum_{t=1}^{T} UC_t * a_t + HC * \max(0, a_t - D_t) + CC * c_t$$
$$s.t.\, a_1 = I,$$
$$a_t \le UL_t,$$
$$\sum_{\tau=1}^{t} a_\tau \le M,$$
$$c_t \le NC.$$

result: 15000 ✗

with Our Method

$$\min_{a_t, A_t, c_t} \sum_{t=1}^{T} UC_t * a_t + HC * A_{t+1} + CC * c_t$$
$$s.t.\, A_1 = I,$$
$$A_t = A_{t-1} + a_{t-1} - d_{t-1},$$
$$a_t \le UL_t,$$
$$A_t + a_t \ge d_t,$$
$$A_t \le M,$$
$$A_{T+1} = 0,$$
$$c_t \le NC,$$
$$d_t \le c_t * CL.$$

result: 251500 ✓

Figure 1. Example of optimization problem modeling when multiple constraints are present. It can be observed that using only prompt engineering techniques, such as 1-shot and Chain-of-Thought (CoT), may lead to some constraint information being ignored in the modeling process (highlighted in red in the correct model). However, by incorporating the question partition module from PaMOP prior to modeling, we achieve correct results without losing key information. (For clarity, we have omitted elements such as 1-shot and parameter definitions without affecting the figure's interpretability.)

guided by $\mathcal{G}_{\text{extr}}$, prompts the LLM to produce the structured elements, which are stored in the root node of the partition tree, as shown in Figure 3. We will use $\mathcal{G}$ to refer to all operations processed by LLMs.

We prompt the LLM to assign a vagueness score to each constraint, which is then stored in the root node as part of the constraint information.

After gathering the information in the root node, we partition the problem iteratively using a tree structure. Real-world problems often decompose into loosely connected subproblems with independent variables and semantic differences. For instance, as shown in Figure 1, the problem can be divided into two parts: train-to-yard and yard-to-ship. To partition the problem, we use two methods (Figure 2): independent set separation at the root node and constraint clustering at subsequent layers. The former separates constraints based on mathematical relationships, while the latter clusters them based on semantic information.

Separating independent sets. Inspired by the work of Gasse et al. [2019], we represent the mathematical relationships within the problem as a bipartite graph $G = (V \cup C, E)$, where V denotes the constants and variables involved in the problem (obtained from t_v), C represents the constraints (obtained from t_c), and each edge in E indicates a high-confidence association between a variable and a constraint. We define this confidence level by extracting keywords from constraints and matching them with keywords from variable descriptions, thereby constructing the graph. We then apply graph search algorithms to separate independent subgraphs from the bipartite graph, designating these as children of the root node.

Clustering constraint sets. For each node to be decomposed, we aim to cluster the constraints into several sets. We define the distance between constraints for clustering purposes. In PaMOP, the distance is defined as the inverse of the similarity $s_{i,j}$ between two constraints, as follows:

$$d_{i,j} = \frac{1}{s_{i,j} + \varepsilon} - \frac{1}{1 + \varepsilon}, \tag{2}$$

where the ε denotes a small fixed value.

We combine three methods to measure the similarity between constraints: contextual relationships, keyword matching, and vector similarity. Constraints are typically extracted in order, so adjacent ones are likely to be similar. Thus, we prioritize higher similarity scores for adjacent constraints.

Keyword similarity is determined by the number of common keywords in the top k of two constraints, where keywords are extracted using the term frequency-inverse document frequency (TF-IDF) method. Word vector similarity is calculated using cosine similarity between GloVe embeddings trained on Wikipedia 2014 [Pennington *et al.*, 2014].

We apply weighted averages of these three similarity measures to different layers of the partition tree. Noise points from clustering are treated as potentially relevant constraints rather than removed. The partitioning process continues iteratively until each leaf node contains either a small number of constraints or highly similar ones.

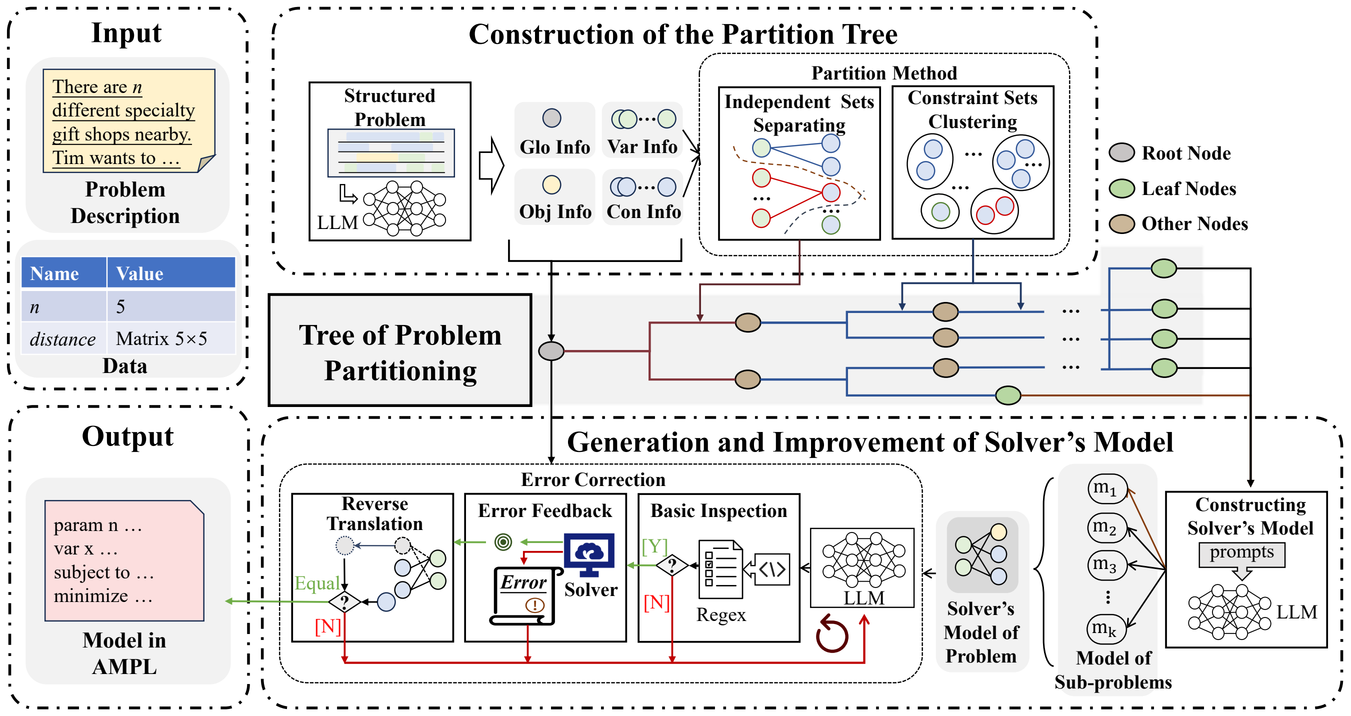


Figure 2. Overview of PaMOP Framework. PaMOP begins by extracting problem information using LLMs. A tree structure then decomposes constraints into simpler sub-problems. Prompt engineering guides the LLM in modeling the leaf nodes, which are subsequently merged bottom-up into a complete model. Finally, the mathematical model undergoes iterative correction through syntax checks, code execution, and reverse translation until error-free.

3.3 Generation and Improvement of Solver's Model

Constructing solver's model. We employ prompt engineering to guide the LLM in modeling all the leaf nodes generated in the previous step. Ultimately, these models are merged into a complete mathematical model in AMPL, which can be run directly when supplemented with the necessary data files. During the modeling of a leaf node node_i, in addition to the constraint descriptions t_c and the constant and variable information t_v required for the model, we also need to provide the global summary g of the problem as input to ensure that the LLM maintains an overall understanding of the problem. The modeling process is given by:

$$m_{c,i} = \mathcal{G}_{\text{mod}}\Big(g, t_v, \bigcup_{j\,\in\,\text{cons}_i} t_{c,j}\Big), \tag{3}$$

where the $cons_i$ means the list of constraints contained in node i, and $t_{c,j}$ means the j-th constraint in t_c.

Motivated by the Chain-of-Thought (CoT) approach [Wei *et al.*, 2022], we introduced a comprehensive process for specifying optimization problems within prompts. Additionally, we developed a set of principles for the task of modeling optimization problems to ensure the stability of outputs generated by LLMs.

When modeling vague constraints, achieving logically coherent results can be challenging, as different constraints may exhibit varying degrees of vagueness in their descriptions. By utilizing the tree structure to decompose the original problem, when modeling nodes containing vague constraints, we can incorporate information from their parent and sibling nodes to aid in the modeling process.

After each leaf node is modeled separately, the formulas will be merged layer by layer from the bottom up into a complete model. Since the constants and variables t_v have been described in advance, there is minimal conflict between formulas modeled at different nodes. Thus, we can directly merge the modeled formulas and finally complete the modeling of the objective function at the root node. At this stage, the overall model can be expressed as:

$$M = (m_p, m_v, m_o, m_c) = \mathcal{G}_{\text{mod}}\left(g, t_v, t_o, m_c\right). \tag{4}$$

This process generates the complete model. We use AMPL [Fourer *et al.*, 1987] for modeling, as it separates the model and data files. Unlike humans, LLMs treat mathematical formulas, modeling languages, and programming languages as different "languages," so we directly generate code in the modeling language instead of formulas.

Error correction. After constructing the model, we debug it by checking syntax with regular expressions and verifying parameters against data files. Following Zhang et al. [2023], we apply LLM-based self-debugging: the model and data files are input into a solver, and errors from infeasible solutions are embedded into prompts for iterative correction until a valid model is obtained.

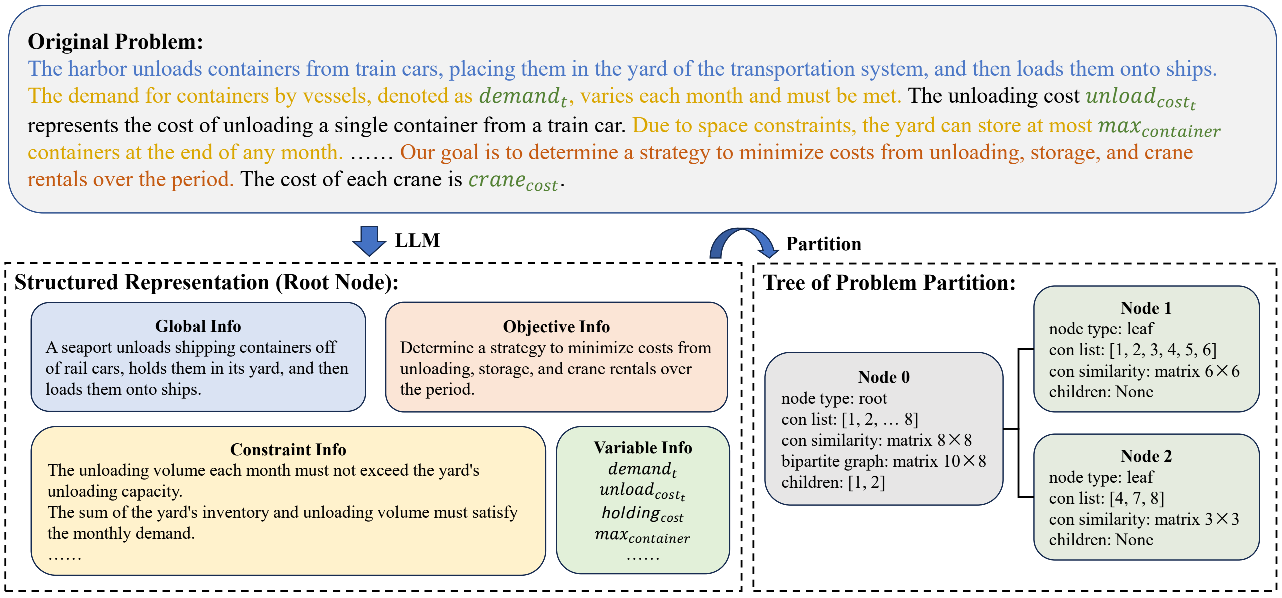

Original Problem:
The harbor unloads containers from train cars, placing them in the yard of the transportation system, and then loads them onto ships. The demand for containers by vessels, denoted as demand_t, varies each month and must be met. The unloading cost unload_cost_t represents the cost of unloading a single container from a train car. Due to space constraints, the yard can store at most max_container containers at the end of any month. …… Our goal is to determine a strategy to minimize costs from unloading, storage, and crane rentals over the period. The cost of each crane is crane_cost.
LLM
Partition
Structured Representation (Root Node):
Global Info
A seaport unloads shipping containers off of rail cars, holds them in its yard, and then loads them onto ships.
Objective Info
Determine a strategy to minimize costs from unloading, storage, and crane rentals over the period.
Constraint Info
The unloading volume each month must not exceed the yard's unloading capacity.
The sum of the yard's inventory and unloading volume must satisfy the monthly demand.
……
Variable Info
demand_t
unload_cost_t
holding_cost
max_container
……
Tree of Problem Partition:
Node 0
node type: root
con list: [1, 2, … 8]
con similarity: matrix 8×8
bipartite graph: matrix 10×8
children: [1, 2]
Node 1
node type: leaf
con list: [1, 2, 3, 4, 5, 6]
con similarity: matrix 6×6
children: None
Node 2
node type: leaf
con list: [4, 7, 8]
con similarity: matrix 3×3
children: None

Figure 3. Example of building the partition tree for an optimization problem.

$$\begin{aligned} \text{solution}(ans, e) &= \text{Solve}(M, \text{data}) \\ M &= \mathcal{G}_{\text{exe}}\,(M, e)\,. \end{aligned} \tag{5}$$

If issues persist, we address logical inconsistencies via reverse translation. Specifically, we extract annotation information that captures the intended meaning of each statement from our analysis of the partition tree and the generation processes. For the constraint under evaluation, we conceal its related annotations to obtain m_c', effectively rendering the model into a partial black box. We then input m_c' into an LLM, which infers the meanings of the constraints based on the complete set of parameters. Another LLM, acting as a language expert, assesses the semantic consistency between the generated sentences and the original ones by comparing them and producing a binary decision (0 or 1). For constraints deemed semantically inconsistent, we forward these—along with the original problem and model—to the LLM, which reinterprets the problem and regenerates the previously problematic constraints to refine the final model.

$$\begin{aligned} t_c' &= \mathcal{G}_{\text{rev}}(m_p, m_v, m_c') \\ \mu &= \mathcal{G}_{\text{comp}}(t_c, t_c'),\;\; \mu \in \{0, 1\} \\ M &= \mathcal{G}_{\text{remod}}(T, m_c, \mu). \end{aligned} \tag{6}$$

4 Experiments

In our experiments, we aim to evaluate the accuracy and execution efficiency of the PaMOP approach in the automated modeling process, particularly for optimization problems with complex constraints. To achieve this, we design a series of experiments, including validating the approach on the NLP4LP public benchmark dataset as well as on several of our real-world problem instances to assess its potential for practical applications. By comparing our approach with existing methods, we further evaluate the performance of PaMOP. Additionally, we conduct ablation studies to assess the contributions of individual modules to the overall performance. Furthermore, we perform a horizontal evaluation of our approach applied to different underlying LLMs further to assess its adaptability and performance in various environments.

4.1 Experimental Setup

In our experimental setup, we tested the system using GPT-4. For these experiments, we set the model's temperature to 0.2 (controls randomness of the model's output) and the maximum number of failed iterations to 5. By default, we input a problem and receive a model file in the AMPL language. To evaluate the accuracy of the model file, we use AMPL to call Gurobi to solve the model and obtain the objective value. To adapt the dataset to the AMPL format, we have preprocessed the dataset's data.json into a data.dat version. When the solver successfully solves the problem without any error, we consider the problem resolved and exit the system.

Finally, we use accuracy to represent the success rate and the execution rate to represent the proportion of model files that can be executed. These two metrics are the primary indicators we focus on. We also employ a compile error rate (CE rate) to capture the percentage of generated programs that fail to compile, which may be due to missing parameters in the modeling process, among other reasons. Additionally, we use the runtime error rate (RE rate) to measure the rate of errors during execution. These errors are mostly due to internal logical errors in the model (such as infeasible models or nonlinear constraints). For instance, when a model is unbounded, it is very likely due to missing constraints.

4.2 Baseline Methods

We conducted a detailed comparison between our method and four approaches based on LLM-enhanced reasoning, including Chain-of-Thought, Progressive Hint, Tree-of-Thought,

	Accuracy ↑	Execution Rate ↑	Compile Error Rate ↓	Runtime Error Rate ↓
Default	25.3%	48.3%	40.2%	11.5%
Chain-of-Thought	28.8%	51.5%	38.7%	9.8%
Progressive_Hint	33.5%	52.3%	34.6%	13.1%
Tree-of-Thought	36.4%	54.4%	35.1%	10.5%
Reflexion	40.3%	69.7%	19.1%	11.2%
Optimus	56.7%	78.4%	11.8%	9.8%
PaMOP (Ours)	**62.3%**	**86.8%**	**7.3%**	**5.9%**

Table 1. Comparison results with baseline methods on the NLP4LP datasets (↑: higher is better, ↓: lower is better).

Methods	Problems					
	Storage	Scheduling	Placement A	Placement B	Mining	HR Allocation
Reflexion	×	×	✓	×	×	×
Optimus	×	×	✓	✓	✓	×
PaMOP	✓	✓	✓	✓	✓	✓

Table 2. Comparison of methods on real-world problem instances.

and Reflexion. In addition, we compared our method with Optimus, which also utilizes LLMs for optimization problem modeling. “Default” represents the default GPT model without any reasoning enhancements; in this setup, we only provided the model with the task requirements to be completed, without adding any additional reasoning prompts or enhancement strategies.

4.3 Benchmark Datasets

The NLP4LP dataset [AhmadiTeshnizi *et al.*, 2024] is collected from optimization textbooks and manuals. It includes problems such as network flow, scheduling, combinatorial optimization, and more. In total, it contains 54 LP problems and 13 MILP problems. Each example contains a description of the problem, the classification of the problem, the dimensions of the input data, and the data file.

We also utilized a custom set of optimization problems derived from real-world scenarios. This dataset includes problems related to storage, scheduling, placement, mining, and HR allocation. It was designed to evaluate the applicability of our approach in practical settings. For instance, the example “Storage” involves 3,795 parameters, 124 decision variables, and 162 constraints.

4.4 Experimental Results and Analysis

The performance of various methods on the NLP4LP dataset is summarized in Table 1, while Table 2 presents results on several real-world problem scenarios encountered during our experiments. Among the different LLM-based reasoning enhancement methods, simpler approaches such as Chain-of-Thought provide limited improvements for the task at hand. Further work demonstrates higher accuracy by introducing preprocessing steps and self-reflection mechanisms.

Our proposed method demonstrates a distinct advantage in performance metrics within the dataset. On the NLP4LP dataset, our model achieved an accuracy rate of 62.3%, which is approximately 5% higher than the state-of-the-art method. In real-world problem scenarios, the Reflexion method was able to solve only one relatively simpler problem, while the Optimus method successfully addressed approximately half of the cases. In contrast, our proposed method was capable of solving all of the problems presented. This significant improvement can be attributed to our approach to detailed problem decomposition and model optimization based on multi-angle feedback. This multi-layered optimization process ensures that the model can reason with greater precision and efficiency when faced with complex optimization problems. The performance would decline considerably if any of these critical components were removed. To further validate and investigate the specific roles of these components and their impact on model performance, we will conduct a detailed analysis and experimental verification in the ablation study.

4.5 Ablation Studies

We use simple pure prompts as a baseline and test on GPT-4, progressively adding components to improve problem-solving performance. The three modes are:

Prompt Only. Specific prompts guide the LLM to output solutions in a given format, with few-shot learning for better task understanding.

Partition + Prompt. The partition component splits the problem into smaller parts, which are modeled separately and then combined.

Partition + Prompt + Correction. An error correction module is added to address infeasible models, performing checks and reverse translation.

As illustrated in Table 3, pure prompts exhibit subpar performance in modeling, successfully tackling only a limited number of problems. The incorporation of a partition module mitigates errors by decomposing larger problems, and the subsequent addition of a correction module further enhances both accuracy and execution efficiency.

4.6 Evaluation of Different Base Models

We also conducted tests using different open-source and closed-source large models as the base. Specifically, we se-

	Accuracy ↑	Execution Rate ↑	Compile Error Rate ↓	Runtime Error Rate ↓
Prompt Only	25.3%	48.3%	40.2%	11.5%
w/ Partition	48.5%	63.4%	26.2%	10.4%
Full	**62.3%**	**86.8%**	**7.3%**	**5.9%**

Table 3. Comparison results of ablation studies (↑: higher is better, ↓: lower is better).

	GPT-3.5-turbo		GPT-4		Llama-3.3-70b	
	Accuracy	Execution Rate	Accuracy	Execution Rate	Accuracy	Execution Rate
Default	5.4%	46.3%	25.3%	48.3%	29.3%	56.2%
Reflextion	11.3%	63.2%	40.3%	69.7%	38.6%	66.4%
PaMOP	24.3%	71.2%	62.3%	86.8%	54.7%	83.2%

Table 4. Comparison results of different models.

lected GPT-3.5-turbo, GPT-4, and Llama 3.3-70b for comparison. As shown in Table 4, all methods benefit from improvements in LLMs, with our approach yielding the most notable performance enhancement. Additionally, we observed that Llama performed well in the pure prompt-based approach; however, as the complexity of the problems increased, its adaptability was slightly inferior to that of GPT-4.

4.7 Insights from Experiments

Imposing strict format requirements. When faced with strict formatting requirements, LLMs often struggle to fully comply with specific standards. For instance, when we require an LLM to embed code in a particular markup format, they may sometimes use different code block styles instead of the one specified. This inconsistency may stem from the model's inherent flexibility and diversity in generating content. Imposing such rigid formats can potentially limit the model's output generation capabilities. In practice, overly strict formatting requirements can interfere with the model's output, as it needs to balance between maintaining the format and ensuring output quality. This, in turn, may affect its performance in logical reasoning and problem-solving tasks, indicating that forcing adherence to fixed formats could diminish the overall capability of LLMs in certain scenarios.

Handling of complex constraints. In practical industrial applications, optimization problems often not only involve a large number of constraints but also exhibit complex logical relationships and a high degree of dynamism. These constraints may include limitations on resources, time requirements, cost control, and the optimization of multiple performance metrics, which may be interdependent or even conflicting. Therefore, solving these optimization problems requires not only a robust algorithmic framework but also precise logical reasoning and flexible strategies to balance and coordinate these complex factors.

However, our method has not been specifically designed to optimize algorithms for these complex constraints; instead, it primarily relies on the inherent logical reasoning capabilities of LLMs. This implies that while LLMs have a certain ability to handle general logical reasoning and constraints, their performance may be limited when faced with high-complexity problems in real-world scenarios.

5 Related Work

Prompt engineering has been widely explored to improve LLMs' performance. Techniques like CoT prompting [Wei *et al.*, 2022] decompose complex reasoning into step-by-step processes, while its extensions [Zheng *et al.*, 2023; Yao *et al.*, 2024; Besta *et al.*, 2024] refine reasoning paths. Knowledge augmentation and feedback interaction [Shinn *et al.*, 2023; Lewis *et al.*, 2020], as well as compositional multi-prompt learning [Zhu *et al.*, 2023], enhance outputs across various tasks. Program-assisted models like [Gao *et al.*, 2023] and systems like NaturalProver [Welleck *et al.*, 2022] focus on reasoning, code generation, and debugging.

In optimization modeling, OptiMUS [AhmadiTeshnizi *et al.*, 2024] employs prompt engineering to construct mathematical models, debug solver code, and validate solutions. Xiao et al. [2023] improve modeling reliability using multi-agent cooperation. These methods advance LLM capabilities for optimization but face challenges in scaling and accuracy when handling complex real-world problems.

While existing work has advanced prompt design and optimization problem modeling, our approach differs in key aspects: 1) we enhance the effectiveness of prompting across different modeling phases; 2) we incorporate problem partitioning and a deeper focus on background information to handle large-scale problems; 3) we use multi-faceted external inputs for model correction and leverage LLMs' code-reading capabilities, leading to more refined outcomes.

6 Conclusion

In this paper, our goal is to model optimization problems described in natural language, while also being capable of handling large-scale problems. We propose PaMOP, an optimization problem modeling framework based on LLMs to achieve this. The core of PaMOP involves decomposing the original problem into several subproblems, guiding the LLM to model these subproblems, and iteratively correcting errors through diverse methods. Experiments demonstrate that PaMOP exhibits excellent modeling capabilities on the NLP4LP dataset and can effectively handle large-scale problems, showing potential for modeling real-world problems in the future.

Acknowledgments

The research is partially supported by National Key R&D Program of China under Grant No. 2021ZD0302900 and No. 2021ZD0110400 , China National Natural Science Foundation with No. 62132018 , "Pioneer" and "Leading Goose" R&D Program of Zhejiang, 2023C01029, the University Synergy Innovation Program of Anhui Province with No. GXXT-2019-024. We also thank Shuli Zeng and the anonymous reviewers for their comments and helpful feedback.

References

[Achterberg, 2009] Tobias Achterberg. Scip: solving constraint integer programs. *Mathematical Programming Computation*, 1:1–41, 2009.

[AhmadiTeshnizi *et al.*, 2024] Ali AhmadiTeshnizi, Wenzhi Gao, and Madeleine Udell. Optimus: Scalable optimization modeling with (mi) lp solvers and large language models. *arXiv preprint arXiv:2402.10172*, 2024.

[Besta *et al.*, 2024] 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 *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 17682–17690, 2024.

[Britz *et al.*, 2013] Wolfgang Britz, Michael Ferris, and Arnim Kuhn. Modeling water allocating institutions based on multiple optimization problems with equilibrium constraints. *Environmental Modelling & Software*, 46:196–207, 2013.

[Burn and McBean, 1985] Donald H Burn and Edward A McBean. Optimization modeling of water quality in an uncertain environment. *Water Resources Research*, 21(7):934–940, 1985.

[Cuchỳ *et al.*, 2024] Marek Cuchỳ, Jiří Vokřínek, and Michal Jakob. Multi-objective electric vehicle route and charging planning with contraction hierarchies. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 34, pages 114–122, 2024.

[Fourer *et al.*, 1987] Robert Fourer, David M Gay, and Brian W Kernighan. *AMPL: A mathematical programming language*. Citeseer, 1987.

[Gao *et al.*, 2023] Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. Pal: Program-aided language models. In *International Conference on Machine Learning*, pages 10764–10799. PMLR, 2023.

[Gasse *et al.*, 2019] Maxime Gasse, Didier Chételat, Nicola Ferroni, Laurent Charlin, and Andrea Lodi. Exact combinatorial optimization with graph convolutional neural networks. *Advances in neural information processing systems*, 32, 2019.

[Gurobi Optimization, 2021] LLC Gurobi Optimization. Gurobi optimizer reference manual, 2021. https://www.gurobi.com/documentation/current/refman/index.html.

[Lewis *et al.*, 2020] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474, 2020.

[Li *et al.*, 2023] Jiaqi Li, Mengmeng Wang, Zilong Zheng, and Muhan Zhang. Loogle: Can long-context language models understand long contexts? *arXiv preprint arXiv:2311.04939*, 2023.

[Li *et al.*, 2024] Tianle Li, Ge Zhang, Quy Duc Do, Xiang Yue, and Wenhu Chen. Long-context llms struggle with long in-context learning. *arXiv preprint arXiv:2404.02060*, 2024.

[Liu *et al.*, 2024] Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173, 2024.

[Mazzoli *et al.*, 2024] Tommaso Mannelli Mazzoli, Lucas Kletzander, Pascal Van Hentenryck, and Nysret Musliu. Investigating large neighbourhood search for bus driver scheduling. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 34, pages 360–368, 2024.

[Middleton *et al.*, 2012] Richard S Middleton, Michael J Kuby, and Jeffrey M Bielicki. Generating candidate networks for optimization: The co2 capture and storage optimization problem. *Computers, Environment and Urban Systems*, 36(1):18–29, 2012.

[Min *et al.*, 2003] Jun-Ki Min, Myung-Jae Park, and Chin-Wan Chung. Xpress: A queriable compression for xml data. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 122–133, 2003.

[Pennington *et al.*, 2014] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[Shinn *et al.*, 2023] Noah Shinn, Beck Labash, and Ashwin Gopinath. Reflexion: an autonomous agent with dynamic memory and self-reflection. *arXiv preprint arXiv:2303.11366*, 2023.

[Shinn *et al.*, 2024] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36, 2024.

[Sitek and Wikarek, 2018] Paweł Sitek and Jarosław Wikarek. A multi-level approach to ubiquitous modeling and solving constraints in combinatorial optimization problems in production and distribution. *Applied Intelligence*, 48:1344–1367, 2018.

[Wei *et al.*, 2022] 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.

[Welleck *et al.*, 2022] Sean Welleck, Jiacheng Liu, Ximing Lu, Hannaneh Hajishirzi, and Yejin Choi. Naturalprover: Grounded mathematical proof generation with language models. *Advances in Neural Information Processing Systems*, 35:4913–4927, 2022.

[Xiao *et al.*, 2023] 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 *The Twelfth International Conference on Learning Representations*, 2023.

[Yao *et al.*, 2024] 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. *Advances in Neural Information Processing Systems*, 36, 2024.

[Zhang *et al.*, 2023] Kechi Zhang, Zhuo Li, Jia Li, Ge Li, and Zhi Jin. Self-edit: Fault-aware code editor for code generation. *arXiv preprint arXiv:2305.04087*, 2023.

[Zheng *et al.*, 2023] Chuanyang Zheng, Zhengying Liu, Enze Xie, Zhenguo Li, and Yu Li. Progressive-hint prompting improves reasoning in large language models. *arXiv preprint arXiv:2304.09797*, 2023.

[Zhu *et al.*, 2023] Xinhua Zhu, Zhongjie Kuang, and Lanfang Zhang. A prompt model with combined semantic refinement for aspect sentiment analysis. *Information Processing & Management*, 60(5):103462, 2023.