Optimization modeling and verification from problem specifications using a multi-agent multi-stage LLM framework

Article *in* INFOR Information Systems and Operational Research · August 2024

DOI: 10.1080/03155986.2024.2381306

CITATIONS
7

READS
541

6 authors, including:

Mahdi Mostajabdaveh
Polytechnique Montréal
13 PUBLICATIONS **103** CITATIONS
SEE PROFILE

Timothy Yu
Simon Fraser University
21 PUBLICATIONS **273** CITATIONS
SEE PROFILE

Zirui Zhou
Simon Fraser University
38 PUBLICATIONS **1,162** CITATIONS
SEE PROFILE

Yong Zhang
Huawei Technologies
105 PUBLICATIONS **2,129** CITATIONS
SEE PROFILE

Optimization Modeling and Verification from Problem Specifications using a Multi-agent Multi-stage LLM Framework

Mahdi Mostajabdaveh [1], Timothy T. Yu [1], Rindranirina Ramamonjison [1], Giuseppe Carenini [2], Zirui Zhou [1], Yong Zhang [1],

[1]Huawei Technologies Canada
[2]University of British Columbia

Abstract

This paper explores the use of Large Language Models (LLMs) in modeling real-world optimization problems. We concretely define the task of translating natural language descriptions into optimization models (NL2OPT) and provide criteria for classifying optimization problems for the NL2OPT task. Our novel multi-agent modeling framework leverages relations identifier agents and a multi-agent verification mechanism, eliminating the need for solver execution. Additionally, we introduce a straightforward and practical evaluation framework, offering a more effective assessment method compared to traditional execution-based evaluations. We have created a unique dataset tailored for optimization modeling, featuring Problem Specifications as a structured representation of optimization problems. Through comprehensive experiments, our study compares our modeling framework with existing LLM reasoning strategies, highlighting their relative effectiveness in optimization modeling tasks. We also perform ablation studies to explore the effect of different components of our modeling framework. Experimental results demonstrate that our multi-agent framework outperforms many common LLM prompting strategies.

1 Introduction

Optimization modeling is the process of representing decision-making problems as mathematical models. A model is a declarative program that abstracts the optimal allocation of limited resources among competing activities, under a set of constraints imposed by the specifications of the problem. It can be applied to different domains such as finance, healthcare, supply chain, and humanitarian operations (Petropoulos et al. 2023). For instance, optimization models can be built to coordinate disaster relief efforts (Balcik, Beamon, and Smilowitz 2008), to allocate medical resources in hospitals (Hulshof et al. 2012), to generate cost efficient production plans (Mostajabdaveh, Salman, and Tahmasbi 2022), or to shape investment strategies (Cornuejols and Tütüncü 2006).

Operations research (OR) projects typically involve many iterations of optimization modeling by OR experts and verification by business owners to develop a model that satisfies all business requirements. This process encompasses two key challenges. First, translating business requirements into a mathematical model is a complex process that requires clear and constant communication with business owners. Second, satisfying the problem specifications demands a deep understanding of both the problem domain and OR techniques. Misinterpretations or misunderstandings of business requirements can lead to a model that fails to adequately address the problem. These issues can lead to increased project costs or even to the failure of the entire project.

Arguably, automating optimization modeling through natural language (NL) descriptions could significantly aid OR experts in efficiently writing and revising optimization models and reducing mistakes associated with limited domain knowledge.

The task of translating natural language problem descriptions into optimization models (or NL2OPT for short) is an emerging challenge (Fan et al. 2024). However, the objective and scope of such a task are yet to be established. Many open questions remain. For example, what should be the goal of the task? Is it the generation of a correct model or an efficient one? What should be the characteristics of the inputs (e.g., structure and content, using technical or non-technical language)? How should we define and evaluate the output of an AI system that performs this task?

For instance, the NL2OPT task input can be presented in many forms, each having a direct impact on the complexity of the task. Table 1 summarizes the characteristics of the NL2OPT task input with respect to description ambiguity and modeling complexity. In this paper, we focus on the task of generating an accurate formulation of an optimization model. The generation of efficient models that can be solved faster, although beneficial, is out of the scope of this paper. In addition, the input and output characteristics of our task differ from previous works in a few ways.

- First, we consider real-world applications of optimization in different application domains. The complexity of the considered optimization problems goes beyond those of linear programming word problems (Ramamonjison et al. 2022b). Note that we focus specifically on the modeling complexity, not the computational complexity of the underlying problems.
- Second, our input consists of a problem summary and a set of problem specifications, both of which are written using non-technical language. In fact, it is common to provide the technical name or category of the optimization problem in the input (AhmadiTeshnizi, Gao,

and Udell 2023).

- Third, there are many possible formats for the output model, which can be classified into two types: (1) mathematical formulations expressed in markup languages (e.g., LaTeX or XML), and (2) optimization code written using a modeling language (e.g., Pyomo, Pulp, and Zimpl). In practice, the mathematical formulation is a convenient and user-friendly format for OR experts, which is often then manually translated into a modeling language as input to the solver. In this study, we consider an end-to-end scenario where the mathematical formulation is generated as an intermediate output and the model code as the final output.

Our aim is to address the question of how LLMs can be used to model real-world optimization problems in practice. Specifically, the main contributions of this paper are as follows:

1. We tackle a practical modeling task that is characterized by a high-level abstraction of the input problem specifications and by the diversity and complexity of the output models. A novel benchmark dataset is created for this new task.
2. We present a multi-agent multi-stage framework that is based on a principled approach to optimization modeling which leverages multiple LLM agents to verify the intermediate mathematical formulation and to generate and review the final model code.
3. We propose a new evaluation framework specifically designed for assessing optimization models. It offers a finer granularity of evaluation compared to execution-based evaluation while avoiding the use of a computationally expensive solver.
4. We test and evaluate different LLM prompting strategies, providing insights about their effectiveness in enhancing accuracy.

The next section provides a review of the current applications and limitations of AI in optimization model building. We then describe the task, present our method, and outline the experimental setup, including the benchmark dataset, evaluation pipeline, and baselines. In Sections 6 and 7, we present our computational results and discuss our findings. Finally, we offer concluding remarks and highlight potential areas for future research.

2 Related Works

Some early works explored the use of natural language processing to augment optimization modeling with an AI assistant. For example, Ramamonjison et al. (2022a) introduced a framework for extracting a model formulation of linear programming word problems (LPWP) using entity recognition and an autoregressive model. The LPWP benchmark, was proposed in (Ramamonjison et al. 2022b) as part of the NeurIPS 2022 competition. This dataset contains 1101 annotated LPWPs from 6 different domains and focused on two shared tasks. These problems typically involved 2 to 3 variables with explicit numerical values in their descriptions. The system presented by Gangwar and Kani (2023) achieved notable results on LPWP by dividing the task into subtasks — initially identifying relations between problem entities and subsequently proceeding with the formulation. Ramamonjison et al. (2023) proposed an interactive system to help OR practitioners convert the mathematical formulation of optimization problems from TeX document format into the solver modeling language.

For many years, one of the main goals of Constraint Programming (CP) has been to align the optimization modeling closely with human language (Hooker 2002; Nethercote et al. 2007). Auto-generation of constraints in CP models has been a subject of study for a while. For example, Beldiceanu and Simonis (2012) and Bessiere et al. (2017) investigated the approach of extracting logical constraints from data. However, only recently with the rapid advancements in LLMs, studies have begun to consider building CP models from NL descriptions of the problem. A notable study by Tsouros et al. (2023) developed a framework using LLMs to convert NL descriptions of problems into CP models. Their framework comprised four main steps; (1) extract the semantic entities of the optimization problem, (2) identify the relations between these entities, (3) formulate the CP model, and (4) translate the problem formulation into code. They also introduced four levels of problem description ambiguity, ranging from unambiguously stating problem name and explicitly tagging model components (e.g., sets, parameters, constraints) to the highest level of ambiguity that resembles non-expert description of the optimization problem. The current study has focused on creating mathematical programming models instead of CP models which require much complex reasoning and higher modeling expertise. More recently, AhmadiTeshnizi, Gao, and Udell (2023) proposed a LLM-based agent designed to formulate and solve mixed-integer linear programming (MILP) problems from their natural language descriptions. They also proposed a dataset with, NLP4LP, with 52 instances each including structured natural language descriptions (SNOP), supervised tests, example data file, optimal value and sample optimal output. However, we argue that their SNOP representation of optimization problems is closer to the technical description of the optimization model than the problem specifications in the application domain. For a comprehensive review of the application of AI methods, including LLMs, in OR, please refer to (Fan et al. 2024).

3 Problem Specification

We introduce *problem specifications* that structure the problem description using only plain text (i.e., without any math symbol or numerical value). Inspired by Sánchez et al. (2021) we divide the specifications into several categories with distinct attributes.

- **Elements:** These encompass the actors within the system, such as people, places, time periods, or machines characterized by specific traits and linked to decisions.
- **Data Parameters:** Data parameters are related to elements and define the input data to the system. Some examples may include production capacity and items value.

Problem Description		Modeling Complexity	
Characteristics	**Description**	**Characteristics**	**Description**
Problem name	Whether explicitly mentioning the name of a known optimization problem.	Individual variables	Whether if variables or parameters are defined individually.
All components	Whether all the sets, parameters, variables, constraints and objectives are defined explicitly with keywords identifying them.	Sets	Whether sets are required to model the problem.
Components relations	Whether the relationship between the model components are defined explicitly.	Number of variables and parameters	The number of variables types and parameters in the model.
Implicit components	Whether implicit model components such as calculations and structural constraints are defined explicitly or not.	Number of constraint	The number of constraint types in the model.
Data abstraction	Whether the data parameters and sets have explicit numerical values or not.	Number of objectives	Whether the model is single objective or multi-objective.
Math symbols	Whether math symbols are used in the description.	Indices with condition	Whether there are conditions on indices for constraints and summations.
OR jargons	Whether OR terms are used or not.	Logical constraints	Constraints that define the logical relations between variables.
Generic or specific	Whether the description is generic or has a specific context.		

Table 1: Criteria for categorizing optimization problems for NL2OPT task input with respect to description ambiguity and modeling complexity

- **Decision Activities:** These are direct actions within the system requiring decision-making, associated with specific elements. These define main variables in the optimization models. Common decision activities include assigning people to tasks, number of goods to purchase, and decision to make a service available.
- **Calculations:** Representing auxiliary variables in optimization problems, calculations derive their values from other decisions and are normally defined by some constraints in the model. For example, an optimization model may calculate the total profit by taking the difference between total revenue and total cost.
- **Objective Criterion:** Outlines the goal or metric to be minimized or maximized by the business owner. Common objective criterion are cost minimization and profit maximization.
- **Requirements:** These are the regulations or limitations that must be adhered to in the optimization problem, defining the conditions for an acceptable solution. Requirements define constraints in the optimization model. Requirements can be implicit and inferred from data parameters (e.g., capacity and demand), or logical relationships between entities (e.g., one variable must be larger than another).

Figure 1 illustrates different problem specifications and their relations to the mathematical model with an example. Problem specifications ensure a clear and precise understanding of the problem, promote effective collaboration between OR experts and business owners during the problem definition phase, guide model development to generate accurate models, and facilitate alignment verification of the generated model with unique needs and goals of the problem owner (Sánchez et al. 2021).

4 Methodology

Our framework employs a multi-agent system architecture (see Figure 2), leveraging LLMs to translate natural language descriptions into mathematical formulations in LaTeX and ultimately generating the model Zimpl code. The framework features many innovative elements, including the development of a Relations Identifier LLM agent which systematically establishes relationships between parameters/-variables and constraints, mirroring the approach of a human modeler. Furthermore, the framework integrates a multi-verifier system with a voting mechanism, ensuring a focused examination of the generated model. An additional novel feature is the effective inter-agent communication to address and rectify errors by implementing two feedback loops (green arrows in Figure 2). These elements collectively signify our technical innovation. We describe each specialized agent in the following.

Relations Identifier: This agent processes each constraint or objective specification individually. It takes all variable and parameter specifications as input and identifies the parameters and variables that should participate in the expression of that constraint or objective.

Modeler: This agent interprets optimization problem specifications and generates a LaTeX-based mathematical model.

Formulation Verifier Agents: After the LaTeX model is generated, a set of verifier agents evaluate its correctness, independently.

- **Consistency Verifier**: Checking the consistency of the generated model, including using the same name for all sets, variables, and parameters in all parts of the model.
- **Relations Verifier**: Use the relations extracted by the identifier to check the generated model including variables and parameter indices, and the participation of variables and parameters in the constraints and objectives.
- **Indices Verifier**: Ensuring that the logical structure of constraints is sound and the indices used in universal quantifiers and summations are correct.
- **Meaning Alignment Verifier**: Verifying that the generated objectives and constraints align with the natural lan-

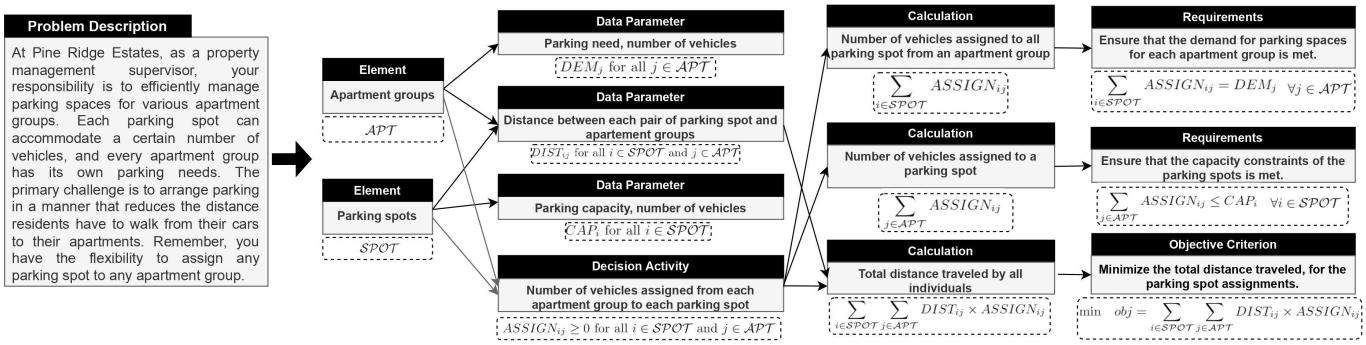

Figure 1: An example of the relations between problem specifications and their relation with the optimization model

guage input by interpreting their expressions' meanings and ensuring they match the related specifications.

Each verifier evaluates the model's correctness by assigning a rating between 1 and 5, with 5 indicating the highest quality and 1 the lowest. Verifiers follow a specific set of rules, basing their scores on the number of rule violations observed in the model. Additionally, they provide detailed feedback, specifying the location within the model and the particular rule violated, to justify their ratings.

Code Converter: This agent translates the LaTeX model into Zimpl code, preserving the structure and intent of the original model.

Code Reviewer: The final stage of the process involves the Code Reviewer Agent, which scrutinizes the generated Zimpl code for errors or discrepancies. Upon detecting an error, the agent provides detailed feedback, including the nature of the error to the Code Converter which is iteratively reapplied.

All the LLM agents in our framework are activated by prompting, please refer to Appendix 9.1 to see the agents' prompt template.

The order in which the LLM agents are applied is illustrated in Figure 2. We initially employ the Relation Identifier agent to identify variables and parameters involved in each constraint or objective. This relational information is then utilized as input for both the Relations Verifier and the Modeler agents. Once the Modeler generates the LaTeX model, the four verifiers evaluate the model's correctness independently, providing ratings and their justifications. The framework computes an average rating from the verifiers to determine the model's correctness. If this average exceeds a certain threshold, the model advances to the next phase. If rejected, a feedback loop is initiated, with the verifiers' reasons forwarded to the Modeler Agent for improvements. After LaTeX formulation is generated and passed the verification, the Code Converter Agent translates the model into Zimpl code. The Code Reviewer Agent then evaluates this code, approving or rejecting it with reasons. Rejected code prompts the Code Converter to revise the code based on the feedback provided.

5 Experiments setup

5.1 Dataset Overview

To gauge the performance of our method in more realistic scenarios, we require a dataset that presents greater complexity than the simple LP problems in the NL4OPT competition (Ramamonjison et al. 2022b) and offers less structured input than those in (AhmadiTeshnizi, Gao, and Udell 2023). Furthermore, due to the scarcity of suitable optimization model resources, we had to construct our own dataset. Table 2 summarizes the characteristics of existing datasets and our new dataset. We employed four operations research experts to create the dataset, who meticulously handcrafted each instance. Each instance comprises: (1) A context providing a case study description of an optimization problem in natural language. (2) the corresponding problem specifications, and (3) the optimization model code in Zimpl. Figure 3 shows an instances of the dataset.

The OR experts based their work on popular optimization problems such as the traveling salesman problem, minimum spanning tree, portfolio optimization, blending problem, job shop scheduling, and supply chain network design, creating complex variants of these problems. Emphasis was placed on writing context-specific problem descriptions, avoiding generic phrases like "bound constraint on all variables" or terms like "limit sum of smallest j". Additionally, special care was taken to abstract data parameters without specifying numerical values (e.g., "the total storage must not exceed the warehouse storage capacity" vs. "the total storage must not exceed 15 squared meters").

In generating the dataset, particular attention was paid to application diversity and model complexity. Specific criteria were established: the number of sets and variables must not exceed 5, the number of parameters must be fewer than 8, and the number of constraint types must be limited to 8. This criterion was selected to maintain a balance in complexity. All problems are single-objective optimizations, varying in type from linear programming and mixed-integer programming to quadratic programming. The dataset contains 70 instances, covering 15 application domains, with each domain represented by at least three problems. These domains span a diverse range, including healthcare and well-being, urban design, human resources, petroleum, and sales.

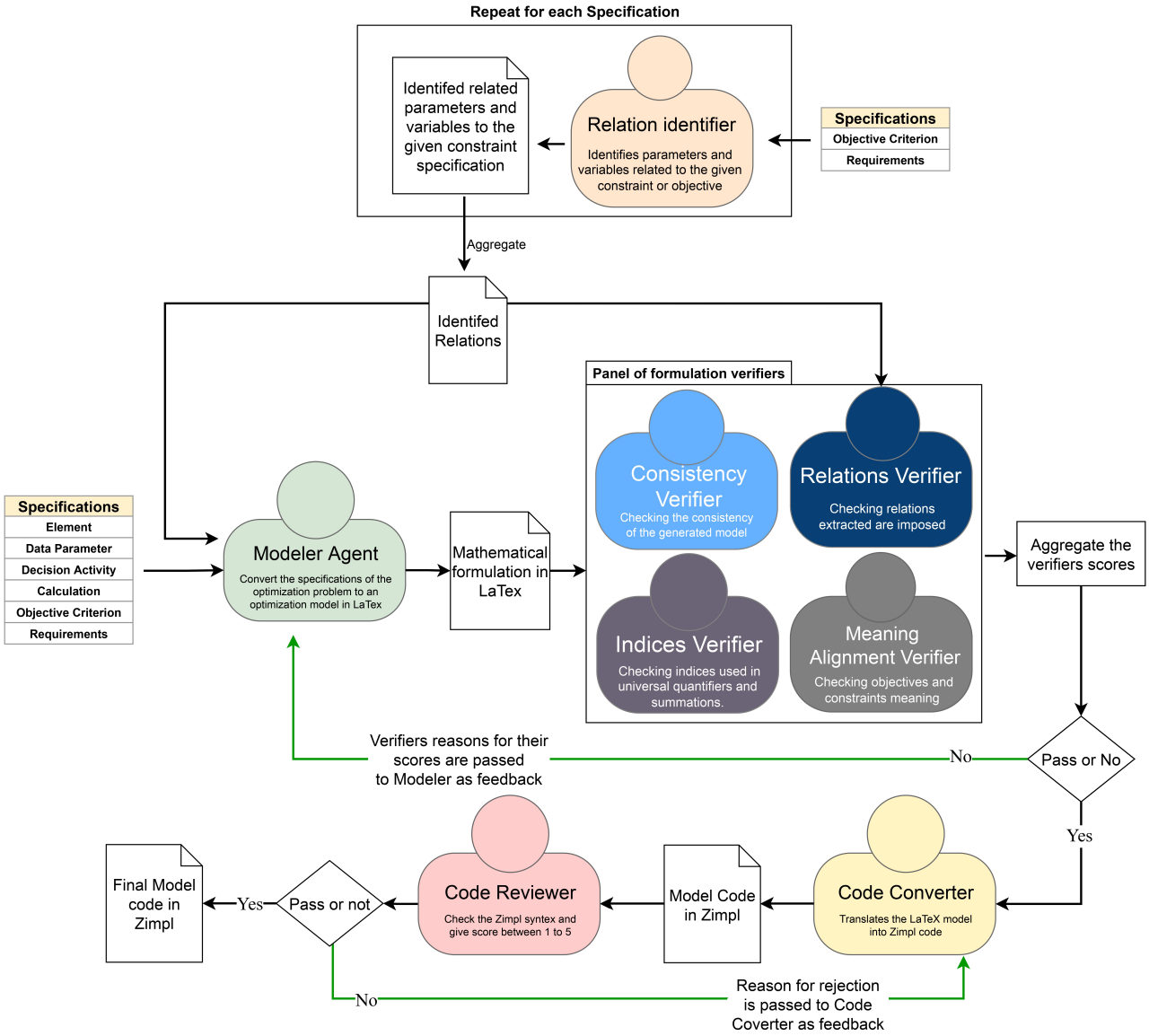

Figure 2: The framework diagram that shows how agents are working together.

Description

Due to the recent surge in COVID-19 cases centered around Yellow Park, as the city's transportation manager, you have been asked to create a plan. The Circle Plaza station, a bustling hub close to multiple tourist attractions, is a region at risk of becoming a super-spreader site due to its high amount of foot traffic. To prevent the spread of the virus from Yellow Park to the Circle Plaza station, the city has decided to temporarily shut down some subway lines. This was proposed to ensure that no subway route exists between Yellow Park and Circle Plaza. While public health is important, shutting down subway lines has its repercussions. There is an impact on city finances. Specifically, impacting ticket sales, staffing, and operational costs. Therefore, you need to make a decision that ensures public safety while also being economically feasible. You have estimated the financial impact associated with closing each track. Your task is to identify which subway routes to shut down, minimizing the overall costs, while ensuring that travellers cannot travel by subway between Yellow Park and the Circle Plaza.

Problem Specifications

- Set of subway stations
- Set of subway routes each connecting two sequential stations
- The cost to shut down a subway route
- Decision to close a specific subway route between a pair of stations
- Decision to include a specific station in the set of stations remain connected to Yellow Park station
- Minimize the cost of closing subway routes to make sure that people cannot travel by subway between Yellow Park and Circle Plaza.
- The Yellow Park station (the Source) is definitely included in the set of stations on the Yellow Park station side, and Circle Plaza station (the Sink) is not.
- If the subway route between two specific stations is closed, then the head station must be included in the set of stations on the Yellow Park side and the tail station is not.

Model Code in Zimpl

```
# Sets:
## Set of subway stations
set STATION;
## Set of subway routes each connecting two sequential stations
set ROUTE[STATION, STATION];
# Parameters:
## The cost to shut down a subway route
param cost[ROUTE];
# Variables:
## Decision to close a specific subway route between a pair of stations
var Z[ROUTE] binary;
## Decision to include a specific station in the set of stations remain connected to
Yellow Park station
var Y[STATION] binary;
# Objective:
## Minimize the cost of closing subway routes to make sure that people cannot travel
by subway between Yellow Park and Circle Plaza.
minimize obj: sum <i, j> in ROUTE do cost[i, j] * Z[i, j];
# Constraints:
## The Yellow Park station (the Source) is definitely included in the set of stations
on the Yellow Park station side, and Circle Plaza station (the Sink) is not.
subto source_sink: Y[first(STATION)] - Y[last(STATION)] >= 1;
## If the subway route between two specific stations is closed, then the head
station  must be included in the set of stations on the Yellow Park side and the tail
station is not.
subto route_cut: forall <(i, j)> in ROUTE do Z[i, j] - Y[i] + Y[j] >= 0;
```

Figure 3: An example of the created dataset. Each instance contains a problem description with a specific context, a list of problem specifications, and ground truth modeling code in Zimpl

Dataset	Description	Modeling Complexity	Model Type	Structured Representation	Output Format
NL4OPT	No problem names, No use of OR jargon, No implicit component, Use of numerical values, Specific context	Individual variables, No sets, Single objective	LP	✓	Symbolic model
NLP4LP	Problem name, No numerical values, All components are mentioned Use of OR jargon Use of Math symbols Generic description	Use sets, Single objective Few variables and parameters, Few constraints	LP, MILP	✓	Model code in Pyomo
Our dataset	No problem name, All components mentioned, No numerical value, No math symbols No OR jargon, Specific context	Use sets, Single objective, Few variables and parameters, Few constraints, Indices with condition	LP, MILP, QP	✓	Model code in Zimpl

Table 2: Existing dataset characteristics

5.2 Approaches Overview

In our experiments, we use different methods for generating optimization model code. We consider five popular prompting approach and seven variation of our method:

- **DESC2MODEL**: The natural language description of the problem is used in a simple prompt to directly generate the optimization model code in Zimpl.
- **SPEC2MODEL**: We provide all problem specifications at once as the input to the LLM with a prompt to directly generate the optimization model code in Zimpl.
- **MULTI-TURN**: Multi-turn prompting, a powerful approach that breaks down tasks into smaller subtasks, has proven effective in code generation (Nijkamp et al. 2022). In our case, each turn constructs a prompt asking the LLM to generate a single component of the optimization model corresponding to the given specification, considering the partial model code generated so far.
- **MULTI-TURN + DESC** Providing only one problem specification at a time may not enable the LLM to comprehend the model as a whole. Therefore, we include the problem description along with a problem specification at each generation turn.
- **MULTI-TURN + SPECS** Similar to previous approach, we provide the entire list of specifications along with a problem specification in each generation turn.
- **MULTI-AGENT-FULL-VERSION** This is the method described in Figure 2 and Section 4. The threshold on the average verification rating for LaTeX formulation acceptance is set to 4.

Methods for ablation experiments:

- **MULTI-AGENT W/O RELATIONS IDENTIFIERS** This method is a variation of the 'Multi-Agent-Full-Version' approach without the Relations Identifier and subsequently the Relations Verifier agent.
- **MULTI-AGENT - W/O {} VERIFIER** This method is a variation of the 'Multi-Agent-Full-Version' without one of the verifiers. Since we have four verifiers, eliminating one at a time results in four different variants of our approach.
- **MULTI-AGENT - W/O CODE REVIEWER** This method is a variation of the 'Multi-Agent-Full-Version' without the code reviewer agent and, consequently, the last feedback loop in Figure 2.

LLMs setup In our experiments, we utilized Code Llama-Instruct (Rozière et al. 2023), a model with 34 billion parameters, built upon the Llama 2 framework for code generation. Specifically, we leveraged Code Llama-Instruct for the following agents: modeler agent, code converter, consistency verifier, and code reviewer. However, Code Llama-Instruct was not proficient in tasks that required reasoning or generating natural language. Thus, we turn to Zephyr-7b-beta (Tunstall et al. 2023) for the following agents: constraint relation identifier, var/param relation identifier, conflict resolver, relations verifier, indices verifier, and meaning alignment verifier. Each agent is initialized with a temperature of 0.1. During each verification feedback step (i.e., panel of formulation verifiers, code reviewer), if the score is below the threshold value, the LLM is passed the prompt once again and asked "This model was scored poorly. Please describe why and how to avoid this issue". The response is passed back and the temperature is increased by 0.05 to facilitate differences in the responses. To prevent a non-terminating loop, the feedback loop is limited to a maximum of three trials. If the score does not exceed the threshold after these trials, we exit the loop and proceed with the process, choosing the output with the highest score to move forward. Also, note that all of the prompts to the LLM contain a one-shot example.

5.3 Evaluation Pipeline

All approaches described in Section 5.2 were evaluated for accuracy across all of the modeling components (i.e., sets, parameters, variables, objectives, and constraints). The generated modeling code was compared with the ground truth model code.

Component matching. The generated components were mapped to a similar component found in the modeling

code using the cosine similarity between the embeddings of each pair of components. Each snippet of code for each component contains a comment that describes the component in natural language and some model code. The natural language comments from both the generated and ground truth models were converted into embeddings using the ALL-MPNET-BASE-V2 sentence transformer (Reimers and Gurevych 2019). The code snippets were converted into embeddings using MATHBERT to capture the mathematical features of the expressions (Peng et al. 2021). The comment embeddings were compared for similarity using cosine similarity and combined (weighted average) with the similarity between the math expression embeddings. This resulted in a similarity measure between every combination of pairs of code snippets between the generated and ground truth model codes. Then, the SciPy LINEAR_SUM_ASSIGNMENT function was used to pair all components to maximize the total similarity score.

Expression equality. The final pairing of components between the ground truth and generated models are then passed to a grammar-based parser (Parr 2013) which was developed to convert the model component code into a symbolic expression. This pipeline maps all generated data (i.e., set, parameter, variable) names to the ones used in the ground truth and evaluates for equality using a custom __EQ__ method. This method converts the math into its symbolic representation that accounts for equality following the order of operations. To describe this using a simple example ($\sum_{i \in S} a[i] + b[i]$ vs. $\sum_{j \in S} b[j] + a[j]$), a symbolic expression is populated that contains information about the set that the variables/parameters are indexed over. The symbolic expressions between the ground truth and generated models are compared. The components that are flagged for inequality or parsing issues are passed to a human expert for review.

Evaluation metric. Accuracy was used to evaluate all approaches for model building. This evaluation method does not consider different ways to formulate the same problem. Rather, the generated expressions for each component must mathematically be identical to the corresponding ground truth component.

6 Computational Results

Baselines Our approach was benchmarked against five methods outlined in Section 5.2: DESC2MODEL, SPAC2MODEL, MULTI-TURN, MULTI-TURN + DESC, and MULTI-TURN+SPEC. These methods were compared to assess their accuracy in generating various optimization model components. Table 3 presents the accuracy of each method on our dataset, providing a clear comparison of their performance in model generation tasks.

Ablation Study We conducted a thorough analysis to assess the impact of novel components of our multi-agent, multi-stage framework on its performance.

We compared seven variations of our method: MULTI-AGENT-FULL-VERSION, MULTI-AGENT W/O CONSISTENCY VERIFIER, MULTI-AGENT W/O RELATIONS VERIFIER, MULTI-AGENT W/O INDICES VERIFIER, MULTI-AGENT W/O MEANING ALIGNMENT VERIFIER, MULTI-AGENT W/O CODE REVIEWER, and MULTI-AGENT W/O RELATIONS IDENTIFIERS. The accuracy of these methods is detailed in Table 4. We observed that the Indices Verifier has the most significant impact on the accuracy of constraints compared to the other verifiers. This impact may be attributed to the reliance of constraint expressions on universal quantifiers and summations. Additionally, the Consistency Verifier considerably contributes to the accurate generation of objectives and constraints, stemming from the tendency of LLMs to use parameters and variables inconsistently.

7 Discussion

Our analysis yielded several key insights.

Structured input significantly impacts accuracy From Table 3, it is evident that SPEC2MODEL outperforms DESC2MODEL significantly. This improvement can be attributed to the structured nature of problem specifications, where each specification clearly defines a single modeling component. Notably, this achievement is realized even though the specifications exclude mathematical symbols, explicit OR keywords, problem names, and are tailored to specific contexts.

LLMs have better accuracy when they have a comprehensive view of the optimization problem In the MULTI-TURN method, we provide only one specification to the LLM at a time, limiting its ability to perceive the optimization problem in its entirety, which in turn affects accuracy. To mitigate this, MULTI-TURN + DESC and MULTI-TURN + SPEC methods were introduced, supplying the LLM with the problem description and the complete list of specifications, respectively. The MULTI-TURN + DESC approach showed the best performance and was chosen as our strongest baseline. Plausibly, the problem description is more effective in presenting the interconnections between model components than a list of specifications, which may be perceived as independent definitions of each component.

Using several task-specific LLM verifiers with a feedback mechanism can increase accuracy The difference between the third and second rows in Table 3 illustrates that having specialized verifiers can significantly enhance the accuracy of NL2OPT for more complex model components (i.e., objectives and constraints).

Limitations While our framework aims to generate correct formulations, it is important to acknowledge that many optimization problems can be modeled in various ways, some more suited for generic optimization solvers. Our primary focus was on correctness rather than optimal performance however exploring the generation of the most efficient models is an intriguing direction for future research. Our dataset was restricted to optimization models with fewer than 5 variables, 8 parameters, and 5 constraints, aiming to limit the complexity of the NL2OPT task. Future efforts could focus on developing methods for accurately creating more complex optimization problems.

				Component Exact-match Accuracy					
Strategy	Multi-turn	Spec input	Desc input	Set	Param	Var	Obj	Constraint	Avg
DESC2MODEL			✓	0.821	0.633	0.448	0.200	0.108	0.529
SPEC2MODEL		✓		**1.000**	0.889	**0.829**	0.586	0.472	0.747
MULTI-TURN	✓	✓		**1.000**	0.832	0.770	0.500	0.426	0.712
MULTI-TURN + DESC	✓		✓	**1.000**	**0.893**	0.789	0.600	0.458	0.751
MULTI-TURN + SPEC	✓	✓		**1.000**	0.881	0.789	0.571	0.463	0.746
OUR APPROACH		✓		**1.000**	0.873	0.786	**0.804**	**0.689**	**0.808**

Table 3: Accuracy for each model component across different prompting strategies.

	Component Exact-match Accuracy					
Strategy	Set	Param	Var	Obj	Constraint	Avg
MULTI-AGENT-FULL-VERSION	1.000	0.873	0.786	0.804	0.689	0.808
MULTI-AGENT W/O CONSISTENCY VERIFIER	1.000	0.855	0.753	0.686	0.608	0.780
MULTI-AGENT W/O RELATIONS VERIFIER	1.000	0.843	0.748	0.771	0.658	0.804
MULTI-AGENT W/O INDICES VERIFIER	1.000	0.833	0.767	0.779	0.581	0.792
MULTI-AGENT W/O MEANING ALIGNMENT VERIFIER	1.000	0.835	0.722	0.729	0.650	0.787
MULTI-AGENT W/O CODE REVIEWER	1.000	0.837	0.752	0.700	0.656	0.789
MULTI-AGENT W/O RELATIONS IDENTIFIERS	1.000	0.869	0.763	0.692	0.658	0.800

Table 4: Accuracy for each model across different variations of our multi-agent multi-stage method.

8 Conclusion and Future Work

This paper introduced a novel multi-agent multi-stage framework for converting natural language descriptions into optimization models, capitalizing on the unique capabilities of each agent to ensure accuracy and coherence. Central to our approach is a voting mechanism among several verifier LLM agents, complemented by a relations identifier agent that discerns high-level relationships between model components. We developed a granular dataset with more realistic optimization problems than existing ones. Our results show that our multi-agent approach surpasses common prompting methods. Furthermore, we conducted an ablation analysis on our multi-agent solution to evaluate the impact of specific components on accuracy. For future work, our framework can be enhanced to focus on generating not only accurate but also efficient optimization models, moving beyond merely ensuring correctness which can be achieved by training models to determine which generated optimization models are suitable for generic optimization solvers. Additionally, our dataset can be expanded to encompass more complex optimization problems or to have more ambiguous problem descriptions as input.

Disclosure Statement

We wish to confirm that there are no known conflicts of interest associated with this publication.

Data Availability Statement

Due to the commercially sensitive nature of the research, the dataset used is not currently available. However, we plan to make the generated optimization modeling dataset public at a later date.

References

AhmadiTeshnizi, A.; Gao, W.; and Udell, M. 2023. OptiMUS: Optimization Modeling Using mip Solvers and large language models. *arXiv preprint arXiv:2310.06116*.

Balcik, B.; Beamon, B. M.; and Smilowitz, K. 2008. Last mile distribution in humanitarian relief. *Journal of intelligent transportation systems*, 12(2): 51–63.

Beldiceanu, N.; and Simonis, H. 2012. A model seeker: Extracting global constraint models from positive examples. In *International Conference on Principles and Practice of Constraint Programming*, 141–157. Springer.

Bessiere, C.; Koriche, F.; Lazaar, N.; and O'Sullivan, B. 2017. Constraint acquisition. *Artificial Intelligence*, 244: 315–342.

Cornuejols, G.; and Tütüncü, R. 2006. *Optimization methods in finance*, volume 5. Cambridge University Press.

Fan, Z.; Ghaddar, B.; Wang, X.; Xing, L.; Zhang, Y.; and Zhou, Z. 2024. Artificial Intelligence for Operations Research: Revolutionizing the Operations Research Process. arXiv:2401.03244.

Gangwar, N.; and Kani, N. 2023. Highlighting Named Entities in Input for Auto-formulation of Optimization Problems. In *International Conference on Intelligent Computer Mathematics*, 130–141. Springer.

Hooker, J. N. 2002. Logic, optimization, and constraint programming. *INFORMS Journal on Computing*, 14(4): 295–321.

Hulshof, P. J.; Kortbeek, N.; Boucherie, R. J.; Hans, E. W.; and Bakker, P. J. 2012. Taxonomic classification of planning decisions in health care: a structured review of the state of the art in OR/MS. *Health systems*, 1: 129–175.

Mostajabdaveh, M.; Salman, F. S.; and Tahmasbi, N. 2022. Two dimensional guillotine cutting stock and scheduling problem in printing industry. *Computers & Operations Research*, 148: 106014.

Nethercote, N.; Stuckey, P. J.; Becket, R.; Brand, S.; Duck, G. J.; and Tack, G. 2007. MiniZinc: Towards a standard CP modelling language. In *International Conference on Principles and Practice of Constraint Programming*, 529–543. Springer.

Nijkamp, E.; Pang, B.; Hayashi, H.; Tu, L.; Wang, H.; Zhou, Y.; Savarese, S.; and Xiong, C. 2022. Codegen: An open large language model for code with multi-turn program synthesis. *arXiv preprint arXiv:2203.13474*.

Parr, T. 2013. *The Definitive ANTLR 4 Reference*. Raleigh, NC: Pragmatic Bookshelf, 2 edition. ISBN 978-1-93435-699-9.

Peng, S.; Yuan, K.; Gao, L.; and Tang, Z. 2021. MathBERT: A Pre-Trained Model for Mathematical Formula Understanding. arXiv:2105.00377.

Petropoulos, F.; Laporte, G.; Aktas, E.; Alumur, S. A.; Archetti, C.; Ayhan, H.; Battarra, M.; Bennell, J. A.; Bourjolly, J.-M.; Boylan, J. E.; Breton, M.; Canca, D.; Charlin, L.; Chen, B.; Cicek, C. T.; au2, L. A. C. J.; Currie, C. S. M.; Demeulemeester, E.; Ding, L.; Disney, S. M.; Ehrgott, M.; Eppler, M. J.; Erdoğan, G.; Fortz, B.; Franco, L. A.; Frische, J.; Greco, S.; Gregory, A. J.; Hämäläinen, R. P.; Herroelen, W.; Hewitt, M.; Holmström, J.; Hooker, J. N.; Işık, T.; Johnes, J.; Kara, B. Y.; Özlem Karsu; Kent, K.; Köhler, C.; Kunc, M.; Kuo, Y.-H.; Lienert, J.; Letchford, A. N.; Leung, J.; Li, D.; Li, H.; Ljubić, I.; Lodi, A.; Lozano, S.; Lurkin, V.; Martello, S.; McHale, I. G.; Midgley, G.; Morecroft, J. D. W.; Mutha, A.; Oğuz, C.; Petrovic, S.; Pferschy, U.; Psaraftis, H. N.; Rose, S.; Saarinen, L.; Salhi, S.; Song, J.-S.; Sotiros, D.; Stecke, K. E.; Strauss, A. K.; İstenç Tarhan; Thielen, C.; Toth, P.; Berghe, G. V.; Vasilakis, C.; Vaze, V.; Vigo, D.; Virtanen, K.; Wang, X.; Weron, R.; White, L.; Woensel, T. V.; Yearworth, M.; Yıldırım, E. A.; Zaccour, G.; and Zhao, X. 2023. Operational Research: Methods and Applications. arXiv:2303.14217.

Ramamonjison, R.; Li, H.; Yu, T.; He, S.; Rengan, V.; Banitalebi-Dehkordi, A.; Zhou, Z.; and Zhang, Y. 2022a. Augmenting Operations Research with Auto-Formulation of Optimization Models From Problem Descriptions. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing: Industry Track*, 29–62.

Ramamonjison, R.; Yu, T.; Li, R.; Li, H.; Carenini, G.; Ghaddar, B.; He, S.; Mostajabdaveh, M.; Banitalebi-Dehkordi, A.; Zhou, Z.; et al. 2022b. NL4Opt Competition: Formulating Optimization Problems Based on Their Natural Language Descriptions. In *NeurIPS 2022 Competition Track*, 189–203. PMLR.

Ramamonjison, R.; Yu, T.; Xing, L.; Mostajabdaveh, M.; Li, X.; Fu, X.; Han, X.; Chen, Y.; Li, R.; Mao, K.; et al. 2023. LaTeX2Solver: a Hierarchical Semantic Parsing of LaTeX Document into Code for an Assistive Optimization Modeling Application. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, 471–478.

Reimers, N.; and Gurevych, I. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.

Rozière, B.; Gehring, J.; Gloeckle, F.; Sootla, S.; Gat, I.; Tan, X. E.; Adi, Y.; Liu, J.; Remez, T.; Rapin, J.; Kozhevnikov, A.; Evtimov, I.; Bitton, J.; Bhatt, M.; Ferrer, C. C.; Grattafiori, A.; Xiong, W.; Défossez, A.; Copet, J.; Azhar, F.; Touvron, H.; Martin, L.; Usunier, N.; Scialom, T.; and Synnaeve, G. 2023. Code Llama: Open Foundation Models for Code. arXiv:2308.12950.

Sánchez, J. M. G.; et al. 2021. Modelling in Mathematical Programming. *International Series in Operations Research and Management Science, Springer*, (978-3): 030–57250.

Tsouros, D.; Verhaeghe, H.; Kadıoğlu, S.; and Guns, T. 2023. Holy Grail 2.0: From Natural Language to Constraint Models. *arXiv preprint arXiv:2308.01589*.

Tunstall, L.; Beeching, E.; Lambert, N.; Rajani, N.; Rasul, K.; Belkada, Y.; Huang, S.; von Werra, L.; Fourrier, C.; Habib, N.; Sarrazin, N.; Sanseviero, O.; Rush, A. M.; and Wolf, T. 2023. Zephyr: Direct Distillation of LM Alignment. arXiv:2310.16944.

9 Appendix

9.1 Prompt templates used for LLM agents

Relations Identifier

You are an Operations Research expert with experience building optimization models for your clients. I need you to help me answer some questions that will help me build the mathematical model. You have an annoying tendency to make up things that have not been asked, so be concise and answer the questions briefly.

I will describe an optimization problem in natural language following 'DESCRIPTION:'. I will then describe one constraint or objective following 'MODEL COMPONENT:'. I need you to understand the optimization problem and help me name the model component after 'COMPONENT NAME:'. I also need you to help me identify the parameters and variables that are used in this component after 'REQUIRED PARAMETERS AND VARIABLES:'. I will give you the description and name of all parameters and variables follows 'PARs/VARs:' You must follow these rules:
1. The component name must be lower case and contain no whitespaces,
2. The list of required parameters and variables must be separated with a semi-colon (;) without any whitespaces,
3. The list of required parameters and variables must consist of only those found in the partial model (DO NOT make up any parameters and variables,
4. The list of required parameters and variables must be exactly as shown in the partial model without any changes.

`{Example}`

DESCRIPTION:
`{Description}`

MODEL COMPONENT:
`{Specification for Constraints/Objective}`

PARs/VARs:
`{Specification for Parameters and Variables}`

COMPONENT NAME:

REQUIRED PARAMETERS AND VARIABLES:

Figure 4: Relations Identifier Prompt Template

Modeler

You are an operations research analyst experienced in creating optimization models given some natural language description. You are tasked with converting the following comments (following SPECIFICATIONS:) into math described using LaTeX. Make sure you follow these rules:
1. Each line of LaTeX must be compilable,
2. Generate one LaTeX math formulation for each line of code,
3. Never change the comments (other than changing # to %),
4. Summation over a set must be defined in the form \sum_{i\in SET1, j\in SET2},
5. Looping over all sets for a constraint must be defined in the form \forall_{i\in SET1},
6. Constraint sense must be one of: \eq, \leq, \geq, \ne,
7. Make sure that you use unique names for sets, variables, and parameters. There must be no overlap in names with any other components,
8. You must define indexed variables or parameters in {}'s - for example, CAP_{SPOT} or $COST_{SPOT, APT}$ where COST is the name, SPOT and APT are the sets it's indexed on.

Here is one example.
`{Example}`

SPECIFICATIONS:
`{List of Specifications}`

OUTPUT:

Figure 5: Modeler Prompt Template

Consistency verifier

You are on a team of operations research experts. You need to verify that a math optimization model (following 'LATEX CODE:') is correct and give it a score of correctness from 1 to 5 and a brief reason for your choice. The latex code must pass these following rules:

- Each line of LaTeX must be compilable,
- Generate one LaTeX math formulation for each line of code,
- Never change the comments (other than changing # to %),
- Summation over a set must be defined in the form \sum_{i\in SET1, j\in SET2},
- Looping over all sets for a constraint must be defined in the form \forall_{i\in SET1},
- Constraint sense must be one of: \eq, \leq, \geq, \ne,
- Check to see if the component names are consistent (i.e., objective and constraint expressions must use the sets, parameters, and variables exactly as written),
- Check if the dimensions of parameters and variables (or the number of indices) are correct,
- Check if variables or parameters are using the incorrect sets,
- Check that each component is correct and consistent,
- Make sure that variable, parameter, and set names are not repeating. One variable cannot have the same name as another component or it will cause issues.

If it is correct and follows all of the above rules, answer after 'SCORE:' a number between 1 to 5 with the following scoring criteria:

- 5 = no errors at all
- 4 = still compilable but there are some minor issues with following the above rules
- 3 = still compilable but does not follow the above rules well
- 2 = not compilable and major issues
- 1 = completely useless model

Also, provide a brief reason for your choice after 'REASON'

```
{Example}
```

LATEX CODE:
```
{Mathematical Formulation in LaTeX}
```

SCORE:

REASON:

Figure 6: Consistency Verifier Prompt Template

Indices verifier

You are on a team of operations research experts. You need to verify that a math optimization model (following 'LATEX CODE:') is correct in that the indices and corresponding sets that in both objective and constraints, the summations (\sum) and forall (\forall) loops are correct and map to the correct sets. You then need to give the model a score of correctness from 1 to 5. The rules for correctness are as follows:

1. Each index in all summations and forall loops in constraints are correctly mapped to sets,
2. Each index in all summations and forall loops in the objective are correctly mapped to sets,
3. There are no extra summations or forall loops in the constraints or objectives,
4. There are no missing summations or forall loops in the constraints or objectives.

If it is correct and follows all of the above rules, answer after 'SCORE:' a number between 1 to 5 with the following scoring criteria:

- 5 = no errors at all
- 4 = one mistake
- 3 = at most 2 mistakes
- 2 = at most 5 mistakes
- 1 = completely useless modeling code

Also, provide a brief reason for your choice after 'REASON'

`{Example}`

LATEX CODE:
`{Mathematical Formulation in LaTeX}`

SCORE:

REASON:

Figure 7: Indices Verifier Prompt Template

Meaning Alignement verifier

You are on a team of operations research experts. You need to verify that a math optimization model (following 'LATEX CODE:') is correct in that it follows the list of specifications that are provided following 'SPECIFICATIONS:'. You then need to give the model a score of correctness from 1 to 5. The rules for correctness are as follows:

1. Each specification comment is mapped to one line of code,
2. The code correctly aligns with each specification comment.

If it is correct and follows all of the above rules, answer after 'SCORE:' a number between 1 to 5 with the following scoring criteria:

- 5 = no errors at all
- 4 = one mistake
- 3 = at most 5 mistakes
- 2 = at most 10 mistakes
- 1 = completely useless modeling code

Also, provide a brief reason for your choice after 'REASON'

`{Example}`

LATEX CODE:
`{Mathematical Formulation in LaTeX}`

SPECIFICATIONS
`{Specifications}`

SCORE:

REASON:

Figure 8: Constraints and Objective Meaning Verifier Prompt Template

Relations Verifier

You are on a team of operations research experts. You need to verify that a math optimization model (following 'LATEX CODE:') is correct in that it follows the relationships that are provided following 'RELATIONSHIPS:'. You then need to give the model a score of correctness from 1 to 5. The rules for correctness are as follows:

1. Each indexed Variable is mapped to sets correctly,
2. Each indexed Parameter is mapped to sets correctly,
3. Each Constraint contains the exact Variables and Parameters (there are no extra or missing Variables or Parameters),
4. Each Objective contains the exact Variables and Parameters (there are no extra or missing Variables or Parameters),
5. Any instance of the incorrect dimensions or incorrect sets in the Variable/Parameter definition is an error.

You must be really strict. We need this model to be perfect. If it is correct and follows all of the above rules, answer after 'SCORE:' a number between 1 to 5 with the following scoring criteria:

- 5 = no errors at all
- 4 = one minor
- 3 = at most 2 mistakes
- 2 = at most 5 mistakes
- 1 = completely useless modeling code

Also, provide a brief reason for your choice after 'REASON'

Here is one example.
`{Example}`

LATEX CODE:
`{Mathematical Formulation in LaTeX}`

RELATIONSHIPS:
`{Identifed relations}`

SCORE:
REASON:

Figure 9: Relations Verifier Prompt Template

Code Converter

You are an operations research analyst and an expert in both LaTeX and ZIMPL code. You are tasked with converting LaTeX code (following LATEX CODE:) into simple and compilable ZIMPL code. Make sure you follow these rules:

1. Each line of ZIMPL must be compilable,
2. Only rewrite the line of code for each component,
3. Never change the comments (other than changing % to #),
4. You have an annoying tendency to hallucinate things. Only generate components that are required for the model described by the specifications,
5. Do not use double-sided constraints; each constraint must have only one comparison. For example, 'a > b > c' must be converted into two constraints 'a > b' and 'b > c',
6. Constraints ZIMPL code must begin with "subto"

Here is one example.
`{Example}`

LATEX CODE:
`{Mathematical Formulation in LaTeX}`

OUTPUT:

Figure 10: Code Converter Prompt Template

Code Reviewer

You are on a team of operations research experts. You are an expert in LaTeX as well as ZIMPL and have been asked to verify that a model has been translated from LaTeX to ZIMPL correctly. The ground-truth model comments in natural langauge are described (following LATEX CODE:) and the ZIMPL model that you must check follows 'ZIMPL CODE:'. The model code within <ZIMPL_CODE> and </ZIMPL_CODE> must pass these following rules:

1. Each line of ZIMPL must be compilable,
2. Do not use double-sided constraints; each constraint must have only one comparison. For example, 'a > b > c' must be converted into two constraints 'a > b' and 'b > c',
3. Constraints ZIMPL code must begin with "subto"
4. Check to see if the component names are consistent (i.e., objective and constraint expressions must use the sets, parameters, and variables exactly as written),
5. Check if the dimensions of parameters and variables (or the number of indices) are consistent,
6. Check if variables or parameters are using the incorrect sets,
7. Check that each component is correct and consistent,
8. When defining the parameters and variables, do not assign any values to it or any bounds. Type definitions (i.e., integer, etc.) is acceptable.

Here is the scoring criteria:
- 5 = perfect, the code abides with all of the rules above
- 4 = at most one mistakes (according to the above rules)
- 3 = at most 3 mistakes (according to the above rules)
- 2 = at most 5 mistakes (according to the above rules)
- 1 = more than 5 mistakes (according to the above rules)

Also, provide a brief reason for your choice after 'REASON'

Here is an example:
`{Example}`

LATEX CODE:
`{Mathematical Formulation in LaTeX}`
ZIMPL CODE:

SCORE:

REASON:

Figure 11: Code Reviewer Prompt Template