GPT-4 Meets Operations Research:

Capabilities and Challenges in Modeling and Solving Business Optimization Problems

BSc Business Analytics

Author: Melis Doga Cevik

Student Number: 14001616

Date: 27 June 2024

Supervisor: Donato Maragno

University of Amsterdam

Abstract

Business optimization and decision-making are essential parts of a business, which so far has relied on human expertise to model and solve mathematical optimization problems. However, the emerging capabilities of Large Language Models (LLMs) offer significant potential in automating this process. This research builds upon previous research done in harnessing LLMs in modeling and solving optimization problems, with a focus on GPT-4. The dataset used in this paper consists of 16 problems, including 4 different optimization classes ((Linear Programming (LP), Integer Programming (IP), Mixed-Integer Programming (MIP), and Nonlinear Programming (NLP)), and varying difficulty. The dataset aims to achieve a real-world representation of business problems, as previous research was criticized for the simplicity of datasets used. This research investigates two pipelines, one with single-step prompting and one with Chain-of-Thought prompting, in order to compare GPT-4's performance in different reasoning environments. The pipelines investigated follow an *end-to-end* setup. The prompting starts from a worded problem description and is carried through GPT-4 until a Pyomo solver code is achieved to reach the optimal solution that a business person is seeking. Through the pipelines, the ability of GPT-4 to mathematically model, and transfer this model into solver code is evaluated. The experiment results reveal that the differences in the two prompting pipelines do not have a statistically significant impact on mathematical modeling abilities of GPT-4. It is observed that although GPT-4 is able to create models and translate them into code, notable gaps remain in obtaining entirely accurate models, with significant failures in constraint formulations. Major setbacks in consistency of the model outputs for the same questions were also observed, which is crucial to improve before leveraging GPT-4 for reliable use in optimization.

Statement of Work

This document is written by Melis Doga Cevik who declares to take full responsibility for the contents of this document. I declare that the text and the work presented in this document are original and that no sources other than those mentioned in the text and its references have been used in creating it. This thesis includes both individual and collaborative work. Chapter 3 on Methodology is predominantly based on group work, with some personal alterations. The LLMs investigated by the research group include GPT-4, Claude 3 Opus, Gemini 1.5 Pro and Mixtral8x22B. The joint results of comparisons between these LLMs as a part of the group deliverable can be found on the Github page. I have not used generative AI (such as ChatGPT) to generate or rewrite text. UvA Economics and Business is responsible solely for the supervision of completion of the work and submission, not for the contents.

Table of Contents

1 Introduction

Mathematical optimization is useful in numerous real-world decision-making scenarios, such as determining the best schedule for loading delivery trucks (Stok, 2022) or devising the optimal distribution of testing and control measures during a global pandemic (Abdin et al., 2023). Generally, companies use their own employee's expertise or hire optimization experts to formulate and solve their problems, which can be extremely time-consuming (Ramamonjison et al., 2023). However, the current developments in the world of AI and LLMs have the potential to change this traditional way of optimizing (Amarasinghe et al., 2023). The ability of LLMs to understand and learn from text, reason, plan, and make decisions at "human-level performance" highlights significant promise in utilizing them for problem-solving (Naveed et al., 2023). Due to this, there are also aims in developing a copilot, like the "Decision Optimization CoPilot (DOCP)" (Wasserkrug et al., 2024), in order to take this advanced vision of the connection between LLMs and optimization to the next level.

Large Language Models (LLMs) have become state-of-the-art Artificial Intelligence (AI) systems that excel in text processing and generation, enabling coherent communication and adaptation to a wide range of tasks. They are trained using an extensive dataset to anticipate the next tokens from a provided input (Naveed et al., 2023)). They are also adapted and fine-tuned to fit many specific tasks and are used in various contexts (Dodge et al., 2020). Thanks to the advancements in this fast-growing industry, LLM tools are made accessible to a broad spectrum of users (Wasserkrug et al., 2024). For example, through the development of ChatGPT (OpenAI, 2022), many people now have access to a chatbot that is good at engaging in "human-like conversation" (De Angelis et al., 2023). This wide availability and groundbreaking abilities of LLMs can be harnessed into many valuable tasks in various industries, ranging from biology and chemistry (Boiko et al., 2023) to teaching and tutoring (Bakas et al., 2024) to the focus of this research, optimization modeling (Yang et al., 2023).

This research aims to contribute by experimenting with the cutting-edge GPT-4 model (OpenAI, 2024a) and evaluate its performance in optimization modeling. This study's central research question is: How effectively can GPT-4 perform in modeling and solving diverse optimization problems? Through some experiments, the research intends to provide valuable insights into GPT-4's OR abilities and highlight the remaining challenges in leveraging it for business decision-making. GPT-4 was picked as the LLM due to its superior performance on benchmark tests (OpenAI, 2024a) as well as its' significantly wide daily usage through the chat interface. Furthermore, in order to evaluate the solving capabilities of GPT-4, Pyomo

(Bynum et al., 2021) was picked as the optimization modeling language due to its ability to model and solve mathematical optimization problems in Python, its range of supported solver packages as well as the researchers' familiarity with it.

The primary objective of this research is to report on GPT-4's performance in optimization problem modeling and analyze specific failure patterns. The research pipelines aim to mimic the interaction of a business person with an optimization copilot. The research has an *end-to-end* setup, starting from a natural language description and leading to a solution that the business person seeks. Most previous research has focused on more simple datasets consisting of only Linear Programming (LP) or Mixed-Integer Programming (MIP) problem types. (Ramamonjison et al., 2023; AhmadiTeshnizi et al., 2023; Li et al., 2023). However, how a single LLM, like GPT-4, performs in many different types of complex optimization problems still needs to be investigated. This research expands on previous research and focuses on several real-world problems diversified by more problem classes, formulation, and constraint types. Furthermore, this research also considers two different pipelines, one more generic and one more advanced, using Chain-Of-Thought prompting (Besta et al., 2024b). The first pipeline was picked to provide a baseline for testing with GPT-4, as a single prompting approach is the most simple way to reach a mathematical model from a problem description. The more advanced pipeline was chosen as previous research has shown improvements in model generation by using this approach (Li et al., 2023). Overall, the evaluation of this complete process aims to cohesively contribute to the performance of GPT4 in OR. This analysis will also help identify further research questions sourced from the success and failure patterns of the experiments, which are crucial to advancing optimization copilot visions.

2 Literature Review

2.1 Background on Large Language Models and GPT-4

The beginning of LLMs can be credited to the experimentation using neural networks and Natural Language Processing (NLP) in the 1950s-60s (Toloka, 2023). LLMs, as we know them today, are built to anticipate the subsequent tokens from a provided input (Naveed et al., 2023). They can be defined as large pre-trained language models, which exhibit "surprising abilities" under challenging tasks. These abilities include in-context learning, instruction following, and step-by-step reasoning (Zhao et al., 2023). Recently, many LLMs have been developed due to their rapid recognition. Some of these are Palm 2 (Google, 2023), Mixtral (Jiang et al., 2024), Gemini 1.5 (Gemini Team, 2024), Claude 3 (Anthropic, 2024), and numerous GPT models by OpenAI. The performance of these various LLMs is greatly related to the quality, quantity, and variety of their training data (Naveed et al., 2023).

The OpenAI LLM models started from GPT-1 (Radford and Narasimhan, 2018), baselining the following Generative Pre-Training (GPT) models with its transformer architecture (Chu et al., 2024), "unsupervised pre-training and supervised fine-tuning" (Zhao et al., 2023). The transformer model is based on a "self-attention mechanism" and allows for parallel processing of input tokens, making it faster (Vaswani et al., 2023). Next, GPT-2 (Radford et al., 2018) was released with an increased parameter scale compared to GPT-1. GPT-3 followed this with larger model parameters and introduced the idea of "in-context learning" (Zhao et al., 2023). GPT-3 demonstrated extreme improvements in NLP tasks compared to previous GPT models. OpenAI then improved GPT-3 to GPT-3.5, which involved "human alignment" and training the model on code data as well as text data, as done previously. Next, GPT-4 (OpenAI, 2023), the focus of this research, was released. GPT-4 showed many improvements in accuracy of answers in reasoning and qualitative tasks compared to GPT-3.5 (Zhao et al., 2023). After GPT-4, further improved models with updated knowledge sources (GPT-4-Turbo) and, most recently, GPT-4o (OpenAI, 2024b) were released.

Several findings in previous research have consistently demonstrated that GPT-4 performs at a higher level than some other LLMs. For example, the research of López Espejel et al. (2023), where performance was evaluated on various datasets of different reasoning categories, as well as mathematical and commonsense questions. In these experiments, GPT-4 outperformed GPT-3.5 and BARD (Manyika and Hsiao, 2023). Additionally, OpenAI (2024a) reports GPT-4 to be more impressive than previous LLMs and other advanced systems available after testing its performance on academic benchmarks. These benchmark tests

include grade school mathematics questions and python coding tasks. Moreover, the insights of Ahmed and Choudhury (2024) on optimization modeling from verbal descriptions also highlight GPT-4s' "superior performance". Due to the enhanced results delivered by GPT-4, both in general tasks and in the context of OR, this research will utilize the GPT-4 API in order to assess its performance in optimization modeling and solving.

2.2 Large Language Models in Optimization Modeling

Previous research has been done to gain insights into the performance of LLMs in optimization tasks. One of the significant contributors was Ramamonjison et al. (2023), with their NL4Opt competition. This competition was about evaluating the ability of NLP techniques to convert written descriptions of mathematical problems into a model. This research was separated into two parts: Identification of optimization problem components and creation of problem formulation, which were evaluated by a micro-averaged F1 score and accuracy, respectively. The winners of the entity recognition task (He et al., 2022) used "ensemble learning with augmentation" (Ramamonjison et al., 2023) to surpass the benchmark results. Furthermore, the winners of the model formulation task found that an overall model generation "at once" is better than separating the formulation into the components of the problem (Gangwar and Kani, 2023). After the competition, Ramamonjison et al. (2023) also did similar tests on GPT-3.5-Turbo, reporting a higher accuracy than the winning models of the competition.

Furthermore, a significant outcome of this competition was the construction of the dataset of linear programming (LP) questions (Ramamonjison et al., 2023). This dataset has since become a commonly used baseline in further research due to its consistency, with examples including studies by Ahmed and Choudhury (2024) and Xiao et al. (2024). However, Ramamonjison et al. (2023) still report that their dataset does not adequately represent the complexity of real-world applications. They highlight that the performance of ChatGPT in modeling more difficult optimization problems still needs to be discovered, leaving room for the focus of this research.

In the research of Ahmed and Choudhury (2024), they compared the performance of GPT-3.5, GPT-4, and a smaller model Llama-2-7B (Touvron et al., 2023) on mathematical model formulation from worded problems. Using the NL4Opt dataset (Ramamonjison et al., 2023), they assessed the performance of these LLMs in zero-shot and one-shot settings, which provides the LLM with either zero or one example when prompting (Ahmed and Choudhury, 2024). They also experimented with fine-tuning the Llama-2-7B model. They report that based on the worded problem descriptions, GPT-4 outperforms the baseline of

Ramamonjison et al. (2023) and also achieves a higher F1 score than GPT-3.5 and fine-tuned Llama-2-7B, with the one-shot setting. Furthermore, Ahmed and Choudhury (2024) mention that although Llama-2-7B performs worse than other models overall, it is better at zero-shot settings due to its smaller size and inability to handle longer texts.

Furthermore, Xiao et al. (2024) also use the NL4Opt dataset as a baseline for their research. However, they also agree with Ramamonjison et al. (2023) and criticize the simplicity of their dataset. Therefore, they build their own dataset (ComplexOR) of 37 problems. This dataset contains more implicit variables and constraints as well as "domain-specific knowledge," reflecting a better representation of real-world scenarios, making it more challenging for the LLMs. Their research focuses on the reasoning abilities of LLMs during optimization modeling. It offers a "Chain-of-Experts (CoE)" methodology, which assigns LLM agents certain expertise in specific tasks within the OR problem-solving process. Their evaluations highlight that CoE outperforms previously researched LLM reasoning ideologies, such as Chain-of-Thought (Wei et al., 2023) and Graph-of-Thought (Besta et al., 2024a). In addition to their default Gpt-3.5-Turbo, they also report that CoE increases the performance of GPT-4 and Claude2, with GPT-4 showing the most improvement in accuracy.

Another study that built their own dataset (NLP4LP) of 52 LP and MILP (Mixed Integer Linear Programming) questions was conducted by AhmadiTeshnizi et al. (2023). This research introduces OptiMUS, an agent driven by an LLM, which uses a structured problem description called "SNOP (Structured Natural language Optimization Problem)". OptiMUS takes SNOP, then produces a mathematical model and solver code, testing its correctness throughout the process and self-editing its code until the problem is solved. It then evaluates the correctness of the solution as well. Their results indicate that the OptiMUS methodology improves the success and execution rate of GPT-4.

Constrastingly, Li et al. (2023) use unstructured problem descriptions. Their research uses a dataset that builds on the NL4Opt dataset (Ramamonjison et al., 2023) by including questions with binary variables and more complex constraint types, like logic constraints. Similar to the previous research, they also focus on mathematical modeling from natural language descriptions but take a three-step approach. First, the decision variables are identified, then objective function and constraints are classified, and lastly, the full MILP model for the problem is formulated (Li et al., 2023). OR experts manually evaluate the quality of each of these steps taken in the formulation process. In addition, the LLMs are fine-tuned on the extended NL4Opt dataset to improve the classification in the second stage. After experimenting with 30 new MILP problems, their results show that the performance of GPT and PaLM significantly improves using the "multi-stage" process, compared to generating a mathematical formulation from a problem description in one step.

Furthermore, Fan et al. (2024) explore the OR pipeline with AI techniques and newer algorithmic methods to model and solve OR problems. They also include a section on performance comparison of different versions of the Llama LLM, with one version fine-tuned on the NL4Opt dataset, seeing its effectiveness on both textbook-level and real-world problems. Their research contributes to the concerns of Ramamonjison et al. (2023), who stated that the performance of LLMs on more realistic problems is not known. They conduct their experiment by feeding the LLMs a prompt: "Write a mathematical model of the following problem description + <Problem Description>" (Fan et al., 2024). Through this prompting, they highlight that the LLMs identify parameters, sets, and decision variables well. However, the overall problem formulation still contains many errors, especially in the constraints. They conclude that LLMs perform quite well on text-book-level problems. However, significant setbacks still remain with real-world applications. In addition, they also find that the fine-tuned version of Llama performs significantly better than non fine-tuned versions. Interestingly, the model sizes of the LLMs did not matter in performance, as a smaller model of Llama outperformed a larger one. Therefore, they highlight that the power of finetuning exceeds the size of the LLM.

In addition to these papers, which researched the ability of LLMs in optimization modeling using many different methodologies and datasets, the vision of a specific copilot for OR has also been experimented with. For example, Amarasinghe et al. (2023) propose an AI Copilot focused on production scheduling problems in business. They use a code-generating LLM called CodeRL (Le et al., 2022) to test their pipeline. Their method consists of a natural language problem description, taken by the LLM, which formulates a coded mathematical formulation for the problem description, which is later solved by the solver. Furthermore, Amarasinghe et al. (2023) also fine-tuned the code-generating LLM by using an approach that divides the problem description into its OR characteristic parts.

Similar to the Copilot introduced by Amarasinghe et al. (2023), Wasserkrug et al. (2024) also foresee a “Decision Optimization CoPilot (DOCP)”. Unlike the work of Amarasinghe et al. (2023), which focuses on production scheduling, they envision this DOCP to model and solve any optimization-related decision-making problems a business person may have. Their requirements for this Copilot include its ability to understand and model the problem, showcasing its understanding of the model type, the data input, and problem variables. In addition, they also state the need for the DOCP to be able to reduce the number of modeling errors the copilot makes and aid the business person in verifying its answers. Lastly, they require the DOCP to model problems in an efficient way such that they are solvable. They experimented with real-world optimization problems using ChatGPT 4. Their results indicate that although ChatGPT showed some promise, it was unable to produce entirely correct and sufficient models, and improvements are needed in

order for it to solve optimization models efficiently.

2.3 Contributions

This research builds upon the foundations set by previous studies in exploring the optimization capabilities of LLMs, focusing on GPT-4. Unlike most prior research that often limited its scope to simpler or a narrow range of optimization problems, this research extends the examination to more complex, real-world problems and a wider range of problem types. These problem types include Linear Programming (LP), Integer Programming (IP), Mixed-Integer Programming (MIP), and Nonlinear Programming (NLP). Inspired by the concerns of Ramamonjison et al. (2023), this research contributes to the testing of LLMs on text-book level and more complex real-world problems. Additionally, this research expands on previous experiments by incorporating a diverse set of constraints—including big M, chance, robust and logic constraints—into the dataset, inspired by the future recommendations of Li et al. (2023). Furthermore, this research integrates unstructured problem descriptions reflective of actual interactions of a business person with an LLM. This aims to enhance the realism in LLM testing environments. In addition, this research also offers a comparison of two prompting pipelines, one with single-step prompting, and one with Chain-of-Thought prompting (Li et al., 2023). These pipelines consist of an *end-to-end* structure, where the natural language description of the OR problem is taken through the LLM until a solver code is written. Furthermore, by using evaluation metrics such as comparing the LLM-generated mathematical models to a "target" formulation (Amarasinghe et al., 2023), this study seeks to quantify the effectiveness of GPT-4 in solving complex optimization tasks. Ultimately, the contributions of this research are intended to advance the groundwork necessary for developing a Copilot, as envisioned by Wasserkrug et al. (2024), thereby enhancing the practical application of LLMs in optimization modeling.

3 Methodology

3.1 Dataset Description

The dataset used to conduct the experiments has been curated with the goal of assessing GPT-4's capabilities of formulating and solving a wide range of real-world optimization problems. It consists of a combination of textbook-level and real-world problems. Previous research has used datasets such as the NL4OPT dataset (Ramamonjison et al., 2023). Due to the shortcomings of this dataset regarding its complexity (Amarasinghe et al., 2023; Xiao et al., 2024), a new dataset was created for more realistic applications. Many real-world use cases require the introduction of integer variables, complex constraints, or even nonlinear terms. The dataset of this research includes these formulations, setting it apart from existing datasets.

While building this dataset, the work of Li et al. (2023) was considered, where they introduced some logic constraints as well as integer variables to create a set of more complex, mixed integer programming problems. Furthermore, the datasets made by AhmadiTeshnizi et al. (2023) and Amarasinghe et al. (2023) also gave insights on how the NL4Opt dataset can be improved. While datasets used across these studies encompass a variety of problem types, each study investigated at most two optimization classes, such as linear programming and mixed-integer programming.

Accordingly, it has yet to be investigated how well a single LLM, such as GPT-4, performs on various kinds of real-world optimization problems characterized by their diversity in terms of optimization classes, difficulty levels, and domains. Considering the characteristics and shortcomings of the datasets mentioned above, the data used in this thesis consists of a more diverse set of mathematical optimization problems. Given the scope of the research, a total of 16 problems were selected. For each of them, the data consists of a natural language problem description, its mathematical formulation, the Python Pyomo code calculating the optimal solution, and any necessary data files containing parameter values. While the natural language description serves as input for the LLM, the mathematical formulations and respective Pyomo code serve evaluation purposes only.

Problems span four types of optimization classes: linear programming (LP), integer programming (IP), mixed integer programming (MIP), and nonlinear programming (NLP). For each of the four optimization classes, four problems of varying difficulty and business domains were selected. More complex problems are mainly characterized by their use of non-trivial constraint types, such as logic constraints or the need for unit conversion of decision variables. The use of such problems was inspired by Li et al. (2023), who

introduced more complexity into the dataset used in the NL4OPT competition by adding four types of logic constraints.

Due to the nature of the use cases, some problem descriptions are formulated explicitly, including the parameter values. In contrast, others, especially larger problems, implicitly define the problem without supplying concrete data in the problem description. Moreover, for a few problems, some parameter values are given explicitly in the description, while parts of the problem requiring larger data inputs are implicitly formulated. Including both explicit and implicit natural language descriptions enhances the realism of the dataset. Business users typically find it more intuitive to describe smaller decision-making problems using concrete values. However, detailing hundreds or thousands of constraints becomes impractical. Finally, a range of optimization problem types was included, such as network flow problems, production planning problems, bin packing problems, and assignment problems. An overview of each problem type, domain, and complexity can be found in Appendix 1. An example of each type of problem can be found in Appendix 2.

Furthermore, the problems were sourced from academic textbooks (Birge and Louveaux, 2011; Bracken and McCormick, 1968; Castillo et al., 2011; Poler et al., 2014), research journal articles (Wasserkrug et al., 2024), university-level lecture materials (problem data: Kurtz, J., personal communication, April 21, 2023; problem logic as seen in Cornuejols and Tütüncü (2006)) and other online sources (MOSEK, nd). Most of these data sources used were readily available online during the training of GPT-4, considering its' training data includes information up to September 2021 (OpenAI, 2023). Therefore, there is a risk that some of the optimization problems were included in the LLMs' training data. Consequently, the problems were altered by reformulating the natural language descriptions. Additionally, in some cases, variables were added, the objective function or constraints were modified, or some parameter values were changed. For the majority of the problems, no Python Pyomo code was available. Hence, it was written manually. These factors help mitigate the risk of using optimization problems as part of this research that the LLM has been trained over. Overall, using a set of such diverse optimization problems helps to more realistically assess whether GPT-4 could be suitable for applications in business optimization settings.

3.2 Experiment Pipelines

Previous literature has experimented with different pipeline designs for the task of generating mathematical optimization problems with LLMs (AhmadiTeshnizi et al., 2023; Fan et al., 2024; Amarasinghe et al., 2023; Li et al., 2023; Ramamonjison et al., 2023). To assess if a more advanced generation pipeline can improve

the LLMs' capabilities, the problems gathered for this study were experimented across two pipelines of different complexities. Both approaches were designed to generate two main outputs to be evaluated: the mathematical model of the given problem and its Python Pyomo code representation. Those outputs were later saved to a repository and assessed by the metrics described further in the document. Pyomo is an open-source Python library for formulating, solving, and analyzing optimization models (Hart et al., 2011). It was chosen for its support of a wide range of problem types, solvers and the researchers' familiarity with its syntax.

3.2.1 Pipeline 1

The idea for the first pipeline was to emulate a simple LLM query, in which the user asks to model their problem and attaches the problem description. A similar approach was taken by Fan et al. (2024) in their experiment. This is also how a business user would likely interact with an LLM chat interface. In this research, system prompts were used through the GPT-4 API, and the problem description was added as the user input. The exact prompt used was:

"Please formulate a mathematical optimization model for this problem. Include parameters, decision variables, the objective function and the constraints in your answer." + problem description

The second sentence was added to the prompt so that the structure of the answer allows for streamlined assessment. Figure 1 shows an example of running this step. Next, with the first LLM response as input, the LLM was asked to generate the Pyomo code corresponding to the mathematical formulation. The prompt used was:

"Please write Python Pyomo code for this mathematical problem. Use sample data where needed. Indicate where you use sample data." + previous LLM response

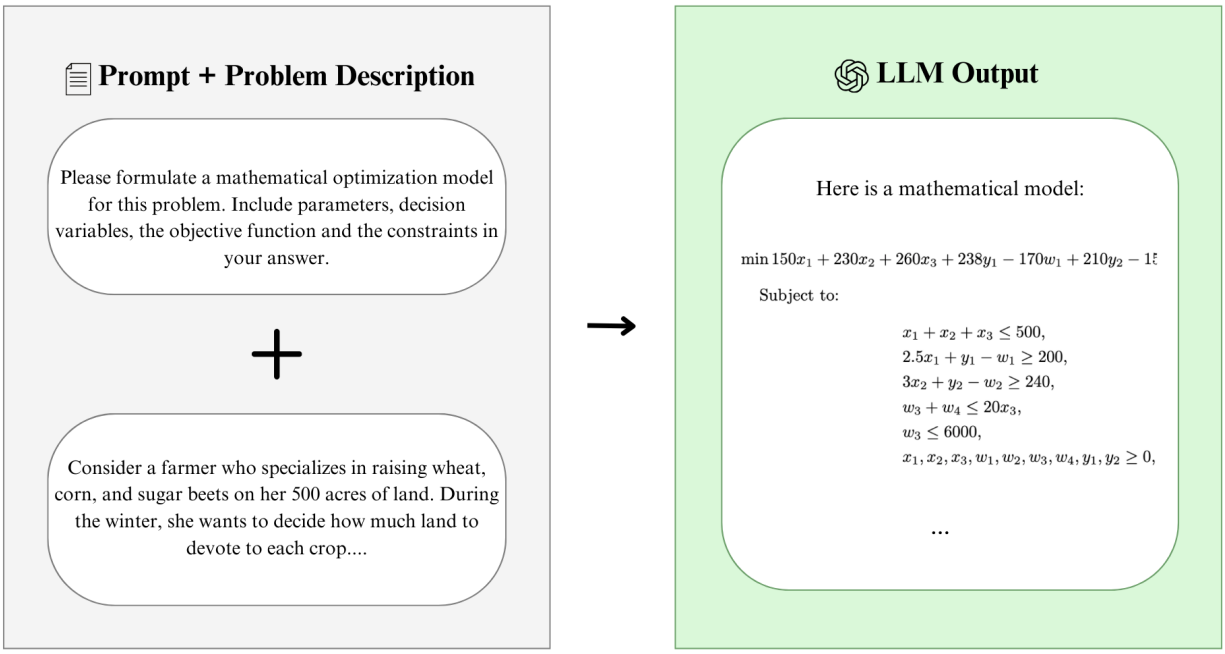


Figure 1: The first step of Pipeline 1, where the system prompt and problem description are given to GPT-4, which then returns a mathematical model.

An example of this step can be seen below in Figure 2. This step was done regardless of the correctness of the mathematical model produced, as even an incorrect model can sometimes achieve the optimal solution, for example, due to the mistakes appearing in inactive constraints. The prompt also includes a statement allowing the model to use sample data where needed. The task of loading the data was considered a software engineering challenge unrelated to the objectives of the experiment. Therefore, the LLM was not expected to provide code to load the required data from files. For the problems requiring data, it was loaded manually, according to the data structure assumed by the LLM. Thanks to this, the problem descriptions could be more realistic without precise data format descriptions.

At this stage, it was also checked if the LLM correctly converted the mathematical model into code. In cases when this step failed, an extra piece of code was written manually, that matches the mathematical model produced by the LLM, to verify the viability of the generated mathematical formulation.

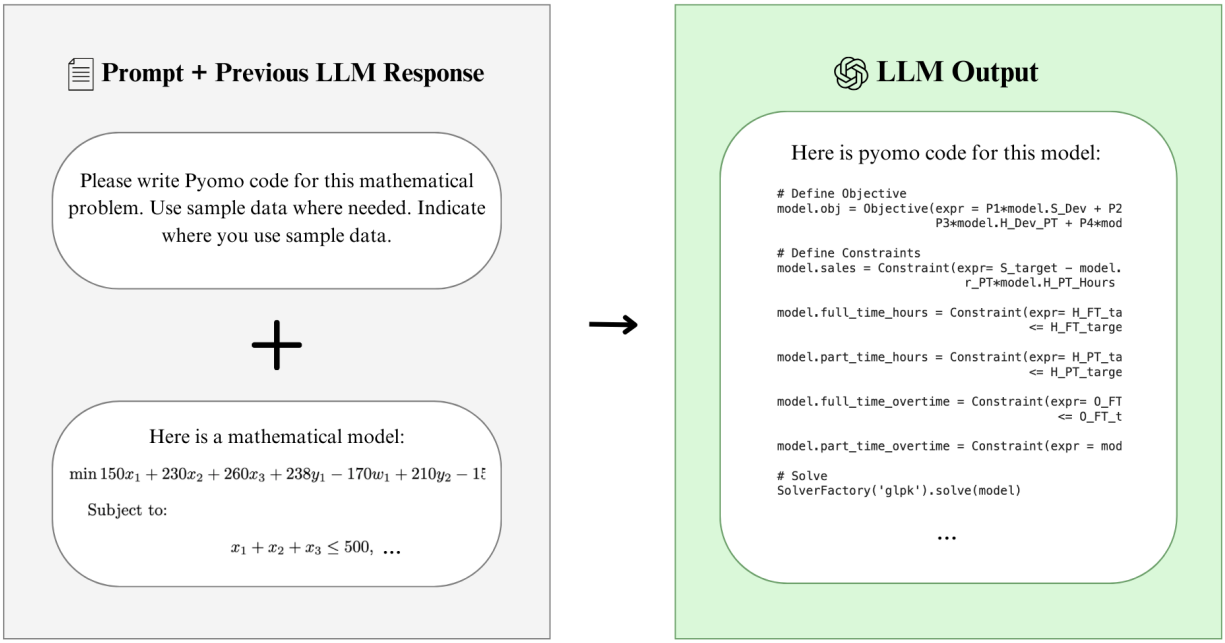


Figure 2: The second step of Pipeline 1, where the system prompt and previous LLM response with the mathematical model are given to GPT-4, which then returns a Python Pyomo code.

3.2.2 Pipeline 2

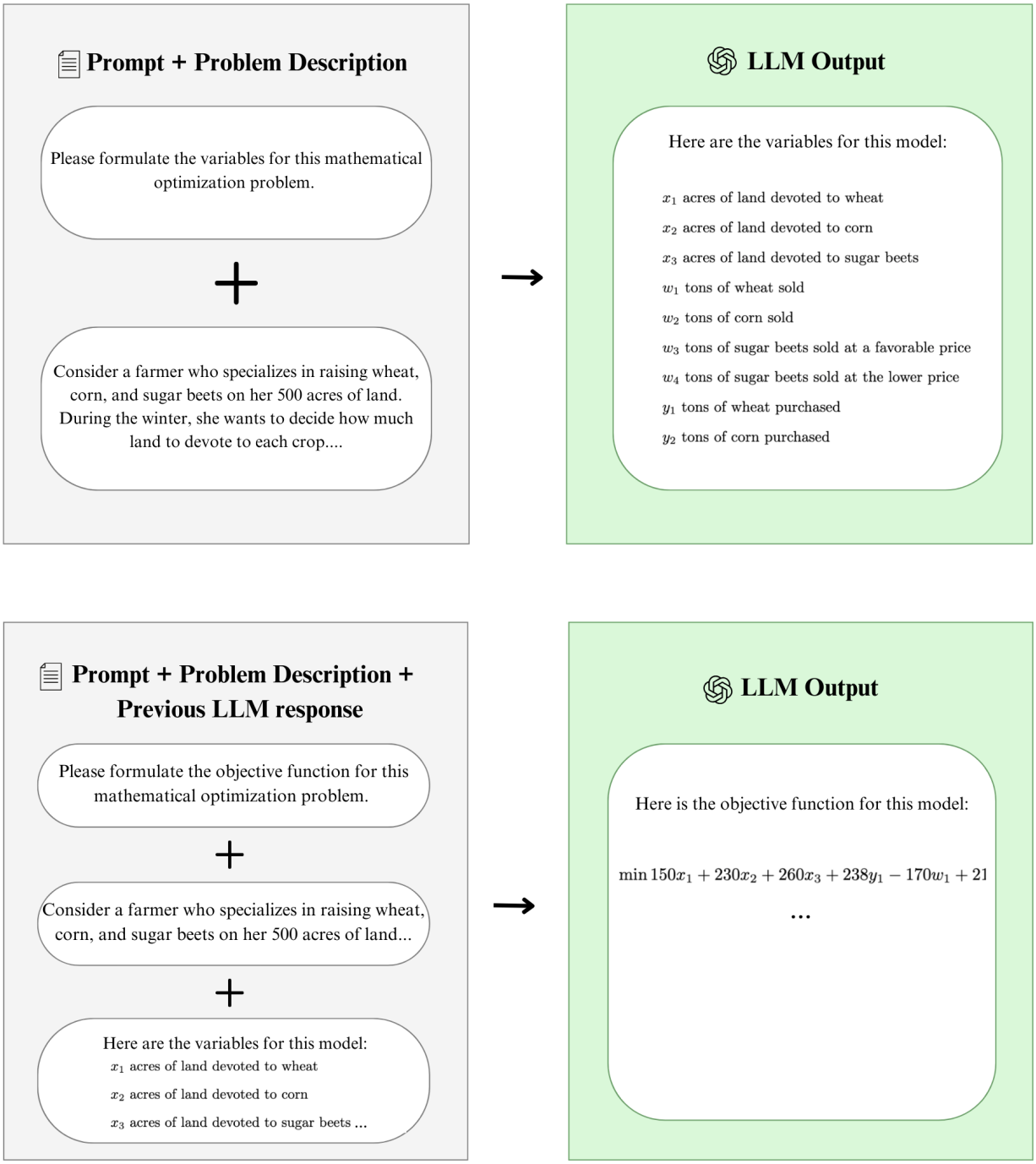
The second pipeline was designed to be a more advanced use of LLMs. The problem description input remained the same as in pipeline 1. However, the task of generating the model was split into three smaller tasks, following a Chain-of-Thought approach. This technique involves breaking down the task into a series of intermediate steps, which helps the model reason through the problem step-by-step (Besta et al., 2024a; Naveed et al., 2023)). Previous research has shown that this multi-step approach improves the performance of LLMs in mathematical modeling (Li et al., 2023). Inspired by this, the execution consisted of asking the LLM for all required variables, then the objective function, and lastly the constraints, with the following prompts:

1. "Please formulate only the variables for this mathematical optimization problem." + problem description

2. "Please formulate only the objective function for this mathematical optimization problem." + problem description + LLM's response to prompt 1

3. "Please formulate only the constraints for this mathematical optimization problem" + problem description + LLM's response to prompt 1 + LLM's response to prompt 2

Such division of work results in smaller subtasks which should be easier for the LLM to solve, leading to an improved final outcome (Besta et al., 2024b). The three responses were then added together manually to form the full mathematical model. This approach can also be seen in Figure 3.



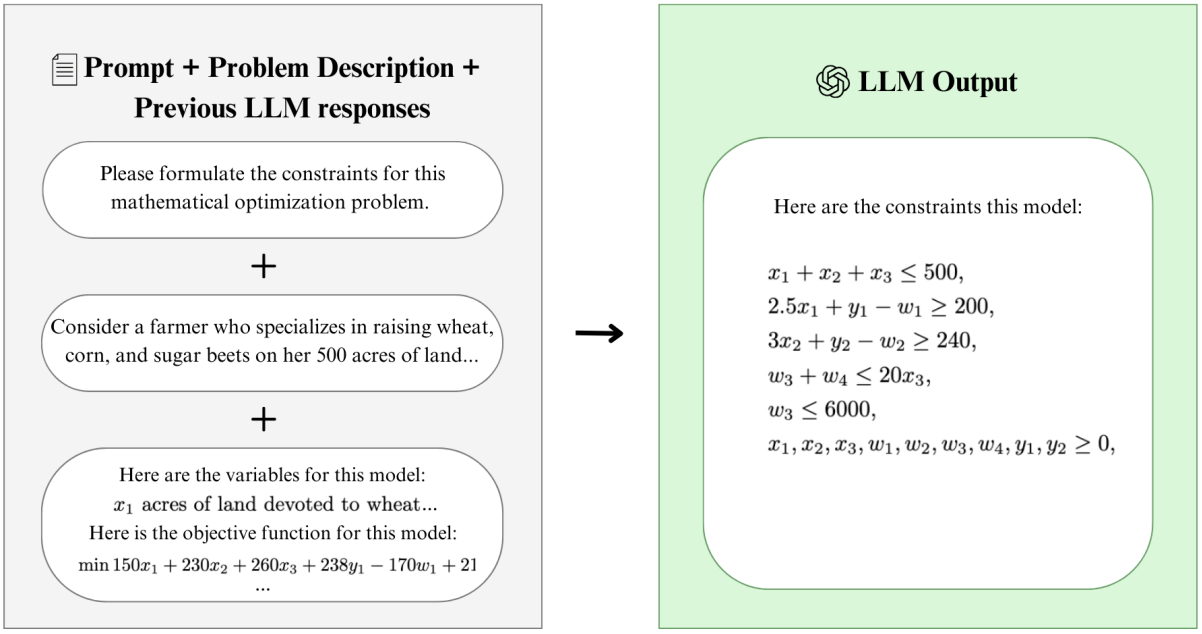


Figure 3: The Chain-of-Thought steps of Pipeline 2, where the prompt and problem descriptions are given to GPT-4, first asking for the variables, then the objective function and lastly the constraints. The previous LLM responses are added to the problem description at each step.

After the steps were combined to form the full model manually, the LLM was asked to generate Pyomo code for the model in the same way as in pipeline 1, as seen in Figure 4. An overview of the pipelines is also shown in Figure 5. Next, the generated outputs were evaluated with the chosen metrics.

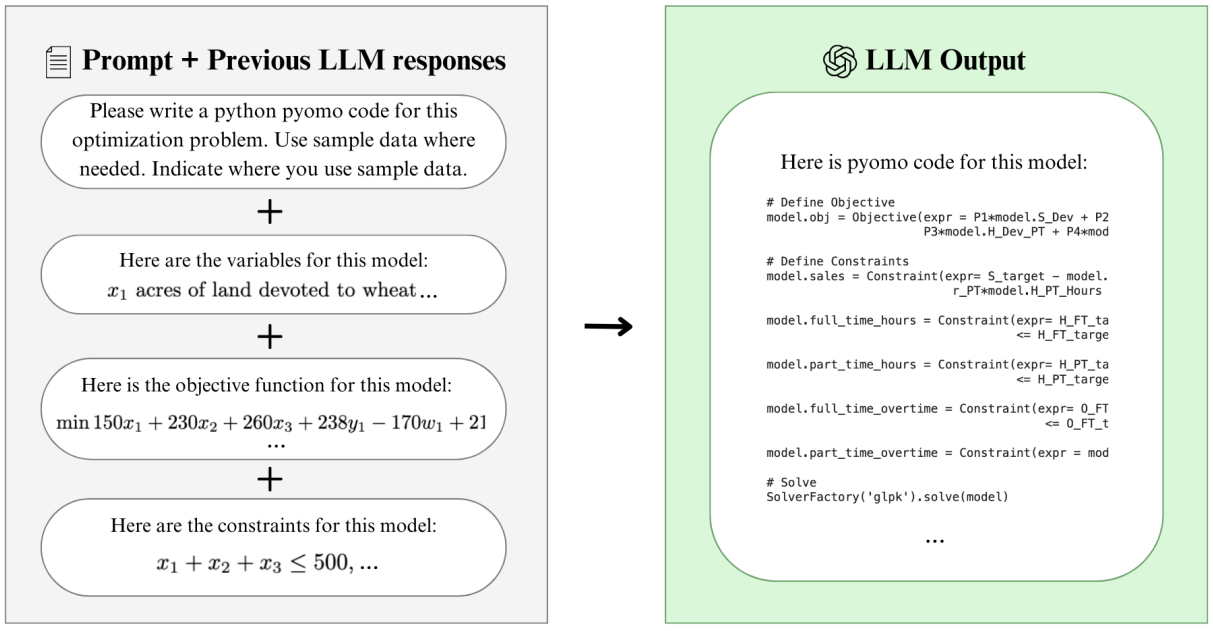


Figure 4: The previous LLM responses of the Chain-of-Thought steps are manually combined and fed into the LLM, with the prompt asking for the code. Then, GPT-4 returns Python Pyomo code.

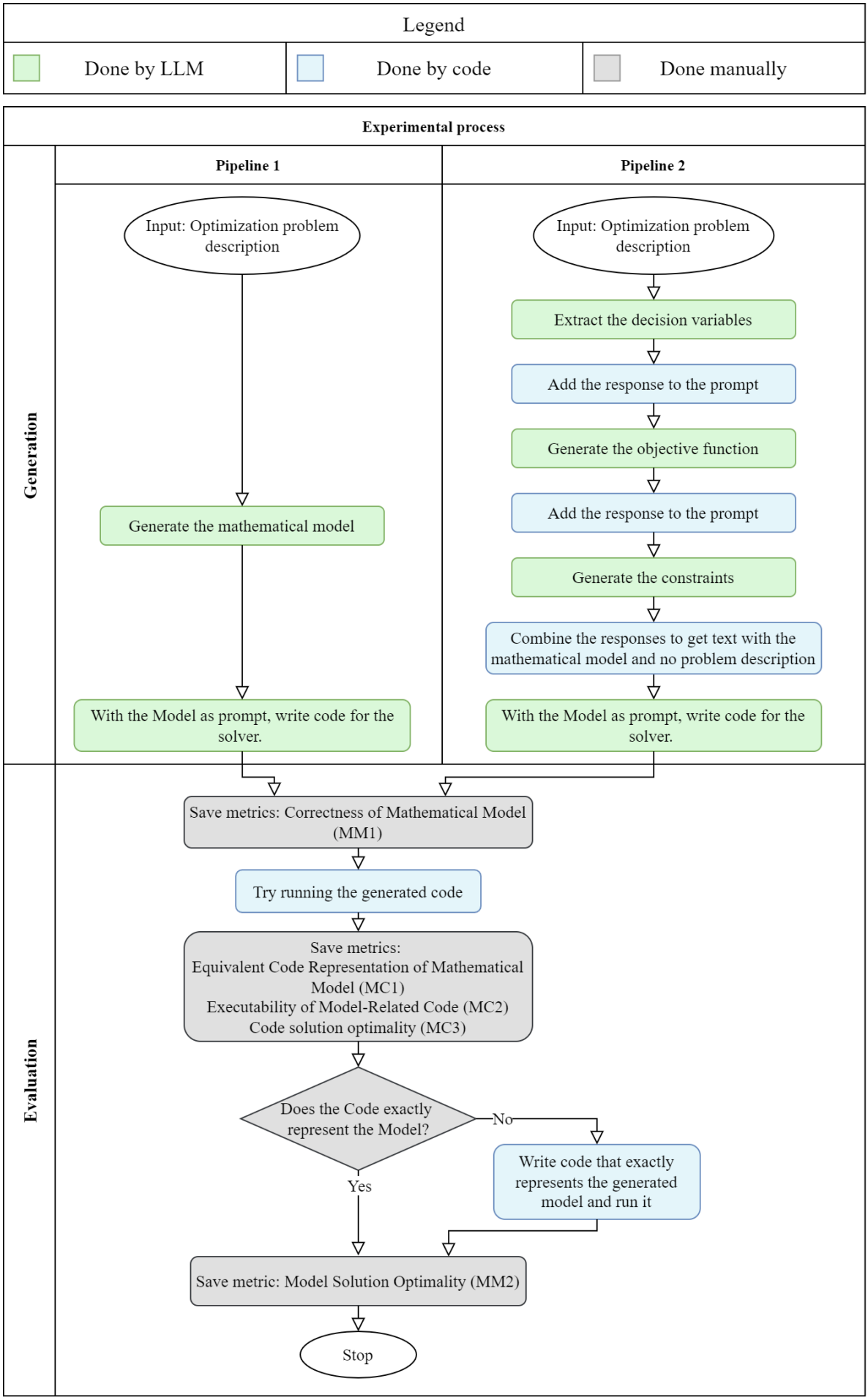


Figure 5: Overview of Pipeline 1 and Pipeline 2.

3.3 Evaluation Metrics

Several metrics were used to quantify the performance of the LLM and correctly translate the problem description to a mathematical model, code, and solution. Generally, the metrics were divided as an evaluation of the generated mathematical model and an evaluation of the code to solve this model. All metrics were recorded manually.

Sarker et al. (2024) analyzed the ability of LLMs to produce code that solves mathematical equations and defined semantic equivalence concepts between mathematical formulas and code. The semantics of two mathematical models are considered to be equal if models always produce the same outputs for all possible inputs. The syntax of a mathematical problem is defined as the sequence of characters that describe the it. Changing the syntax of a mathematical problem while preserving its semantics is called a syntactic transformation. Thus, the problem description, reference mathematical formulation, and reference code for a given problem are semantically equal but syntactically different. As a result, the metrics were measured using the principle of analyzing and comparing semantics between outputs and references. Examples of this are text-based formulations of constraints in the generated mathematical formulation or wrongly referenced Python functions in code that are still considered to be correct if the semantics are equal to the reference. The metrics evaluating the mathematical model are labeled MM1-3, and those for the generated code are labeled MC1-3.

3.3.1 Mathematical Model Evaluation Metrics

Mathematical Model Correctness(MM1):

This metric represents the LLMs' ability to produce an entirely correct mathematical representation of the underlying problem, as described by Li et al. (2023) and Ramamonjison et al. (2023). The mathematical model generated by the LLM and the reference solution were compared to achieve the semantics. To quantify this, the semantic equivalence of the parameters, variables, the objective function, and constraints of the solution generated by the LLM and the reference solution were measured. The equivalence is measured as a binary metric:

$$\text{MM1P}(M,R) = \begin{cases} 1 & \text{if the parameters of the mathematical model } M \text{ semantically match the} \\ & \text{parameters of the reference solution } R, \\ 0 & \text{otherwise.} \end{cases}$$

$$\text{MM1V}(M,R) = \begin{cases} 1 & \text{if the variables of the mathematical model } M \text{ semantically match the} \\ & \text{variables of the reference solution } R, \\ 0 & \text{otherwise.} \end{cases}$$

$$\text{MM1O}(M,R) = \begin{cases} 1 & \text{if the objective function of the mathematical model } M \text{ semantically matches the} \\ & \text{objective function of the reference solution } R, \\ 0 & \text{otherwise.} \end{cases}$$

$$\text{MM1C}(M,R) = \begin{cases} 1 & \text{if the constraints of the mathematical model } M \text{ semantically match the} \\ & \text{constraints of the reference solution } R, \\ 0 & \text{otherwise.} \end{cases}$$

$$\text{MM1}(M,R) = \begin{cases} 1 & \text{if MM1P}(M,R),\ \text{MM1V}(M,R),\ \text{MM1O}(M,R),\ \text{MM1C}(M,R) = 1, \\ 0 & \text{otherwise.} \end{cases}$$

By definition, correct alternative formulations to the reference solution were also considered valid. In pipeline 1, the metric was measured after the first output. In pipeline 2, the four parts of the metric were measured at the according output step. The reason for this is that the model possibly changed parameters, variables, objective function, or constraints at any of the steps, thus leading to ambiguous conclusions.

Model Solution Optimality (MM2):

This metric measures if the objective value produced by the mathematical model of the LLM was optimal, considering the case where the generated mathematical model had additional inactive constraints or different variable domains. Depending on the problem instance, these changes could still lead to an optimal objective value. However, because the semantics of the formulations would be different, this optimal solution would not have been captured by MM1. The objective value was checked and multiple optimal solutions that lead to the same objective value were allowed in solution evaluation. Thus, Model Solution Optimality is defined as:

$$\text{MM2}(M,R) = \begin{cases} 1 & \text{if the objective value of the generated mathematical model } M \text{ and reference} \\ & \text{solution } R \text{ were equal,} \\ 0 & \text{otherwise.} \end{cases}$$

Model Consistency(MM3):

In addition to the accuracy of the generated mathematical model, the consistency of outputs was also of interest. To measure the consistency of LLMs, the outputs were paired as a Cartesian product and the number of mathematically equal output pairs was divided by the total number of output pairs (Raj et al., 2022). Thus, we achieve the solution consistency metric through:

$$MM3 = \frac{1}{n(n-1)} \sum_{\substack{i,j=1 \\ i \neq j}}^{n} I(M_i = M_j)$$

, where n = 3 is the number of runs executed for a given problem, M is the ith generated mathematical model to a given problem and I(Mi= Mj) is an indicator function for the semantic equivalence.

3.3.2 Pyomo Code Evaluation Metrics

Code Equivalence to Model (MC1):

This metric represents the LLMs' ability to apply a syntactic transformation to its generated mathematical formulation. This metric indicates if the LLM transformed its generated mathematical formulation into code without changing its semantics. Notably, it does not assess the correctness of the syntax of the code but of the mathematical formulation within it. More formally, this metric is defined as

$$\text{MC1}(M,C) = \begin{cases} 1 & \text{if the semantic meaning of the generated mathematical model } M \text{ and the conversion} \\ & \text{into code } C \text{ were agreeing,} \\ 0 & \text{otherwise.} \end{cases}$$

Code Executability (MC2):

The code executability metric measures how well the LLM could use the chosen coding tools (Python

and Pyomo) to arrive at a solution after a mathematical formulation was provided. As was proposed by Xiao et al. (2024), the generated code was tested for its executability, regardless of the solution value. It is formally defined as:

$$\text{MC2}(C) = \begin{cases} 1 & \text{if the code executed without errors,} \\ 0 & \text{otherwise.} \end{cases}$$

If an error was produced after a solution was calculated (e.g. while displaying the calculated solution), this metric is still evaluated to 1. This error may indicate that GPT-4 struggles with printing Pyomo outputs. However, this research focuses on the ability of GPT-4 to effectively use Pyomo, rather than software engineering abilities such as printing.

Code Solution Optimality (MC3):

Finally, the objective value that resulted after a successful code execution was evaluated. Data was replaced to fit the original problem instance. This metric measures the LLMs' ability to autonomously arrive at the correct solution that was associated with the original problem description, as was seen in AhmadiTeshnizi et al. (2023). The objective value was checked and multiple optimal solutions that lead to the same objective value were allowed in solution evaluation. The formula of this metric is:

$$\text{MC3}(C) = \begin{cases} 1 & \text{if the optimal solution and objective value produced by the code generated by the} \\ & \text{LLM was correct,} \\ 0 & \text{otherwise.} \end{cases}$$

By thoroughly combining these metrics in the evaluation, a precise description of GPT-4s' ability to solve optimization problems was achieved. It is important to note that all metrics, with the exception of MM3, were measured for each experiment run and later aggregated, MM3 was measured once for each problem.

4 Results

A total of 48 experiments were conducted, consisting of 16 distinct problems, each tested across three separate runs. The experiment results will be reported on different levels, consisting of the overall performance of the pipeline across the whole dataset and performance on different problem types. Performance results on an individual problem basis can be found in Appendix 3. Each experiment was run three times per problem, and an average accuracy score was reported. The problem types are Linear Programming (LP), Integer Programming (IP), Mixed Integer Programming (MIP) and Non-Linear Programming (NLP).

4.1 Mathematical Model Evaluation

Across both pipelines, the ability of GPT-4 to translate a natural language description into a mathematical model was evaluated with MM1: Correctness of Mathematical Model, as well as the correctness of the optimal solution this model produces with MM2: Model Solution Optimality. The consistency of these outputs was also measured with MM3: Model Solution Optimality. Table 1 below shows the results obtained for all problem types across the two different pipelines.

Type	Parameter Correctness	Variable Correctness	Objective Function Correctness	Constraint Correctness	Mathemetical Model Correctness	Model Solution Optimality	Model Consistency
			Pipeline 1				
LP	0.58	0.58	0.33	0.08	0.00	0.17	0.00
IP	0.92	0.33	0.50	0.25	0.17	0.25	0.00
MIP	0.75	0.42	0.42	0.00	0.00	0.17	0.00
NLP	1.00	0.67	0.75	0.08	0.08	0.25	0.00
ALL	**0.83**	**0.50**	**0.48**	**0.10**	**0.06**	**0.21**	**0.00**
			Pipeline 2				
LP	0.75	0.92	0.83	0.25	0.17	0.25	0.00
IP	1.00	0.25	0.25	0.00	0.00	0.17	0.08
MIP	0.83	0.25	0.25	0.17	0.08	0.08	0.00
NLP	0.83	0.75	0.58	0.25	0.08	0.33	0.17
ALL	**0.84**	**0.54**	**0.48**	**0.17**	**0.08**	**0.21**	**0.06**

Table 1: The table shows the results for evaluation metric MM1: Mathematical Model Correctness and its components, MM2: Model Solution Optimality and MM3: Model Consistency. The results are average success rates per problem type and for the whole dataset, across Pipeline 1 and Pipeline 2, reported to 2 d.p.

The correctness of parameters, variables, objective function, and constraints were measured as a part of the overall mathematical model correctness (MM1). For parameter correctness, GPT-4 achieved a success

rate above 0.8 across both pipelines. The rare instances of failures were primarily due to missing parameters. Furthermore, there were occasional errors in parameter interpretation, where some parameters got assigned incorrect values.

When defining variables, GPT-4's performance was 50% and 54% for pipelines 1 and 2, respectively. The primary failure trend observed in variable definitions was in binary variables. Therefore, the variable correctness of IP and MIP problems is much lower than LP and NLP problems (see Table 1). Among the 24 collective runs conducted on IP and MIP problems, GPT-4 failed to define indicator variables in 13 instances for pipeline 1 and 12 instances for pipeline 2, each surpassing 50% of the experiments. This highlights a weakness in GPT-4's capacity to comprehend and formulate binary variables accurately. Additionally, GPT-4 also struggled with variables including a dependence on time, for example, in problem MIP-4, where it consistently did not define a variable required for water release at time t for a water network optimization problem. Moreover, GPT-4 also struggled with defining variables that involved positive or negative deviations, like in problem IP-4 when working hours of employees could deviate.

Moreover, the formulation of the objective function emerged as a notable area of weakness for GPT-4 in optimization modeling, achieving a success rate of 0.48 across both pipelines. The correct understanding of the objective was present in most of the experiments, using maximization and minimization functions correctly. However, problem-specific mistakes were common in objective function formulation, including errors stemming from previous mistakes in parameter and variable definitions. Furthermore, instances of creation of previously undefined variables in the objective function were also observed, indicating inconsistencies in model formulation. Additionally, in a network optimization problem, the objective function occasionally focused solely on specific paths rather than considering all available alternatives, further highlighting areas where GPT-4's performance fell short in achieving accurate objective function representations.

The most significant failure point of GPT-4 in mathematical modeling was constraint formulations. They were formulated correctly 10% and 17% of the time in pipelines 1 and 2, respectively, across all the experiments (see Table 1). These low success rates highlight substantial weaknesses in GPT-4's capability to comprehend and formulate constraints within optimization problems. Predominantly, failures were due to missing or incorrect constraints and errors stemming from previously inaccurate variable definitions. Furthermore, GPT-4 frequently failed to accurately integrate complex constraints, leading to complete scoring penalties even if only a single incorrect constraint was present. For example, it struggled with using big M constraints when they were required to activate certain investment scenarios in problem IP-2. In addition, GPT-4 also failed at formulating robust constraints, for example, in LP-2, where a robust drug

production problem required different amounts of varying ingredients in the drugs. Furthermore, another observation regarding constraint formulation is that GPT-4 achieved a success rate of 0.00 in all MIP problems in Pipeline 1 and across all IP problems in Pipeline 2 (see Table 1). This indicates that none of the constraints formulated across these problems were correct. This could be due to the inability of GPT-4 to accurately define binary variables, which cascades into constraint formulation errors, as binary variables are essential for constructing the expected constraints in these optimization problems.

Overall, the previously mentioned failures in formulating components of optimization problems result in a low overall Mathematical Model Correctness, with rates of 6% for pipeline 1 and 8% for pipeline 2. Specifically, pipeline 1 produced only two enitrely correct formulations out of 48 experiments, corresponding to one run in problem IP-1 and problem NLP-2. Furthermore, pipeline 2 yielded four correct formulations, one each in problems LP-1, LP-3, MIP-1, and NLP-2. These results highlight that GPT-4 frequently struggles to produce entirely accurate mathematical models. A significant contributing factor to these poor outcomes is GPT-4's difficulty in constraint formulation. In pipeline 1, approximately one-third of the failures in overall model correctness were solely due to errors in constraint formulation, even when the rest of the model was accurately constructed. A similar pattern was observed in pipeline 2, where one-quarter of the failures were attributed to constraint formulation mistakes. Moreover, the dataset included both explicitly and implicitly stated problems. This distinction in problem description did not significantly impact the correctness of the models generated by GPT-4. However, it was observed that GPT-4 tends to formulate mathematical models implicitly, regardless of whether the problem description is explicit. For instance, in pipeline 1, for problem MIP-3, despite the problem description explicitly constaining certain numbered parameters, GPT-4 consistently ignored these and formulated the entire problem implicitly in every run. This tendency shows a characteristic of GPT-4 in optimization modeling. Although implicit formulations are not incorrect, they require data to be inputted by the user at a later stage again, which is unnecessary if the user already had explicitly inputted the data with the description.

Furthermore, the optimality of the solutions produced by the mathematical models formulated by GPT-4 was evaluated using the MM2 metric. Both pipelines exhibited an identical Model Solution Optimality rate of 0.21 (see Table 1). Notably, several problems were consistently solved incorrectly across all runs and both pipelines, indicating that GPT-4 uniformly failed to generate correct answers for these specific questions. Figure 6 illustrates the successes and failures observed in Model Solution Optimality for each problem across both pipelines. Specifically, eight problems, namely LP-2, LP-4, IP-4, MIP-2, MIP-3, MIP-4, NLP-3, and NLP-4, were never correctly solved by the mathematical models generated by GPT-4. The complexities present in these problems, such as robust constraints, logic constraints, time-dependent vari-

ables and constraints, and lengthy, domain-specific problem descriptions, likely contributed to the observed failures. Furthermore, Model Solution Optimality (MM2) was higher than overall Mathematical Model Correctness (MM1), as depicted in Table 1. This can be attributed to the presence of inactive constraints and specific constraints that do not alter the solution in their correct formulation. For instance, in NLP-1, GPT-4 mistakenly applied the production capacity constraint on a weekly basis despite the problem description specifying a daily basis requirement. Although this constraint was inaccurately formulated, it did not impact the final optimal solution derived by the model.

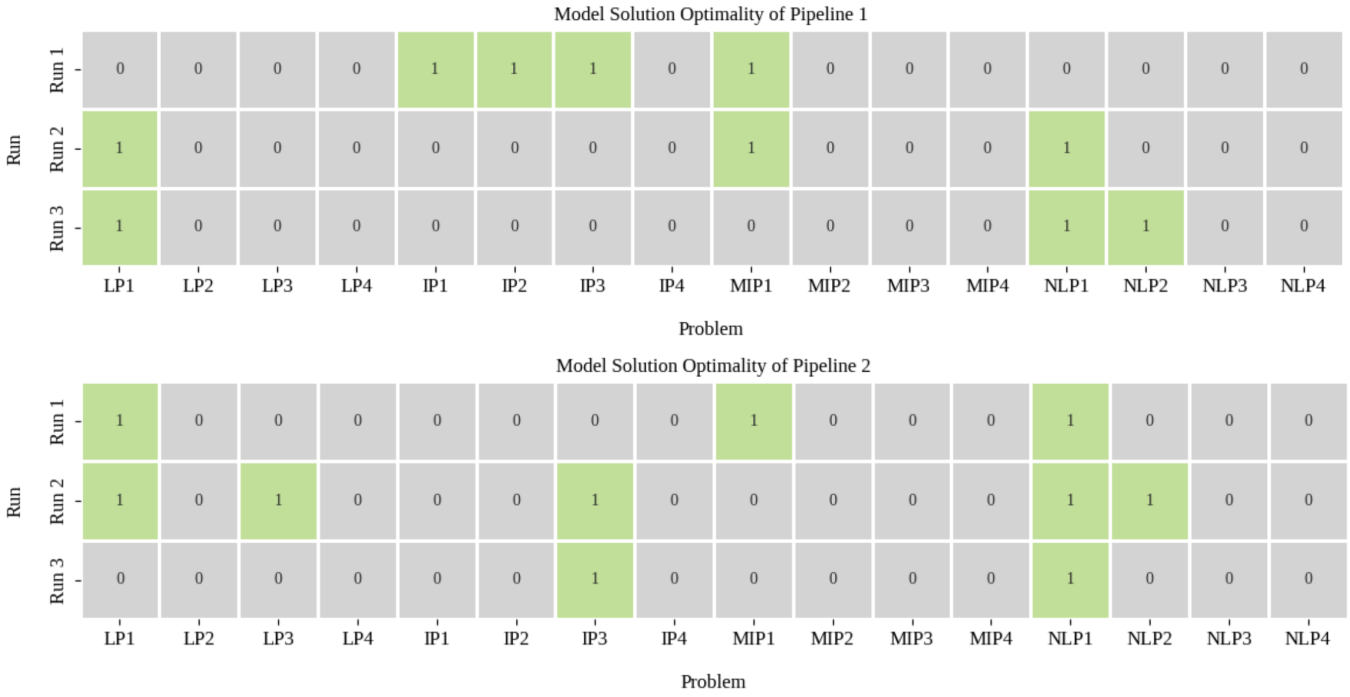


Figure 6: This figure shows the Model Solution Optimality across all three runs of all problems for Pipeline 1 and Pipeline 2. The green areas indicate correct and optimal solutions (1), the grey areas indicate incorrect solutions (0).

The experiments also assessed the consistency of GPT-4's outputs using MM3, revealing notable discrepancies in the answers produced for identical questions. Consistency within pipeline 1 was notably absent, with no identical outputs observed across all runs for any of the problems. In contrast, in pipeline 2, there were three pairs of consistent answers, achieving a 6% consistency rate across the dataset. Each of these consistent answers was observed in IP-3, NLP-1, and NLP-2 (see Table 1). The consistency observed in NLP-1 and NLP-2 could be attributed to the clear and explicit wording of their problem descriptions. For instance, the objective function in NLP-1 was specifically defined as "49000 times X1 minus X1 squared plus 30 times X2 minus two times X2 squared," leaving little scope for varied interpretations or formulations by GPT-4. However, it is essential to note that the consistency in these NLP problems predominantly resulted from repeated errors in constraint formulation. Therefore, this form of consistency should not be

mistaken for successful performance.

Overall, the performance scores of GPT-4 on mathematical modeling show minimal difference between the two different pipelines, as illustrated in Table 2. One notable difference was seen in Model Solution Optimality scores across different problem types. Pipeline 1 was superior in IP and MIP problems, whilst pipeline 2 outperformed pipeline 1 in LP and NLP problems (Appendix 4). However, both pipelines overall achieve identical Model Solution Optimality (MM2) scores of 21%, across the whole dataset. Similarly, Mathematical Model Correctness (MM1) and Model Consistency(MM3) scores show comparable results between the two pipelines, with a slightly higher score observed for pipeline 2 in comparison to pipeline 1. Statistical analysis using p-values suggests that the differences observed in these metrics are not significant, indicating no difference in effectiveness between the two prompting approaches.

Evaluation Metric	**Pipeline 1**	**Pipeline 2**	***p*-Value**
Mathematical Model Correctness (MM1)	0.06	0.08	0.70
Model Solution Optimality (MM2)	0.21	0.21	1.00
Model Consistency (MM3)	0.00	0.06	0.09

Table 2: The table shows the success rates across the whole dataset of all Mathematical Model Evaluation Metrics (MM1, MM2 and MM3) and the statistical significance of the performance difference between Pipeline 1 and 2, using a p-value evaluated at a significance level of 0.05.

4.2 Pyomo Code Evaluation

In order to evaluate the performance of GPT-4 in optimization modeling, the correct representation and executability of the code, as well as the solution it produced were also quantified. The *glpk* solver in Pyomo was used to solve LP, IP and MIP problems. The *knitro* and *couenne* solvers were used to solve NLP problems. Table 3 below shows the code evaluation metric results obtained for all problem types across the two different pipelines.

The ability of GPT-4 to translate a mathematical model to Pyomo code exhibited a success rate of 60% in pipeline 1 and 58% in pipeline 2. In both pipelines, the inaccuracies consisted predominantly of omissions and additions of model components. In pipeline 1, 8 out of the 19 errors were due to omissions of elements that were previously defined in the original mathematical model. Conversely, 7 out of these 19 errors were attributed to the introduction of new variables and constraints that were not present in the initial model. In pipeline 2, the distribution of errors showed a similar pattern but with different proportions: 16 out of 22 mistakes were omissions, while 4 out of 22 were unnecessary additions.

Furthermore, the executability of the Pyomo code generated by GPT-4 was assessed, revealing that only 56% of the code in pipeline 1 and 54% in pipeline 2 was capable of running. This demonstrates that just

Type	Code Equivalence to Model	Code Executability	Code Solution Optimality
		Pipeline 1	
LP	0.50	0.50	0.08
IP	0.83	0.67	0.17
MIP	0.67	0.58	0.08
NLP	0.42	0.50	0.25
ALL	**0.60**	**0.56**	**0.15**
		Pipeline 2	
LP	0.75	0.58	0.25
IP	0.58	0.75	0.08
MIP	0.50	0.42	0.00
NLP	0.50	0.42	0.33
ALL	**0.58**	**0.54**	**0.17**

Table 3: Shows MC1: Code Equivalence to Model, MC2: Code Executability and MC3: Code Solution Optimality. The results are success rates per problem type and for the whole dataset, across Pipeline 1 and Pipeline 2, reported to 2 d.p.

over half of the generated experiments resulted in operational code, regardless of the model's feasibility. A significant portion of the errors—30% in pipeline 1 and 32% in pipeline 2—stemmed from the use of incompatible syntax elements, such as if statements or functions like *max(x,y)*. Pyomo's syntax does not support these elements in the objective functions and constraints, rendering the code non-executable despite the semantic correctness of the formulations in matching the model. Other errors were generally due to other Pyomo-specific coding mistakes, such as failing to define a ConcreteModel or improperly formatting double-sided constraints as required by Pyomo's syntax. Additionally, in cases involving nonlinear problems, GPT-4 defaulted to using *glpk* as the solver in approximately 50% of the experiments. Although this was considered a software engineering matter and not penalized under the evaluation metrics, it represents a recurring issue in GPT-4's capacity to appropriately select solvers for generating executable Pyomo code.

Lastly, the solution produced by the code GPT-4 returned was also evaluated, achieving success rates of 0.15 and 0.17 in pipeline 1 and 2, respectively. No consistent pattern in code solution failures was observed, indicating a variety of underlying errors rather than a singular systemic issue. These failures are primarily linked to issues in Code Equivalence to the Model, Code Executability as well as Mathematical Model Correctness. In addition, concerning the feasibility of the solutions generated, only 2 out of the 48 experiments conducted in each pipeline resulted in infeasible outcomes. This suggests that while the majority of solutions were technically feasible, their alignment with the intended mathematical model was often compromised, thereby affecting the overall correctness of the solution generated by GPT-4.

5 Discussion

Contrary to its promise in prior studies (AhmadiTeshnizi et al., 2023; Ahmed and Choudhury, 2024; Ramamonjison et al., 2023; Xiao et al., 2024), GPT-4's performance in this study was suboptimal. Its model formulation and solver code generation for optimization problems fall short of the requirements for use as an AI Copilot by business professionals (Wasserkrug et al., 2024). This conclusion holds across both tested prompting pipelines. Compared to previous research, this unsatisfactory performance can be attributed to the heightened complexity of some of the problems used and the design of the pipelines implemented in this research. Numerous instances of modeling errors were observed, alongside the generation of inefficient and unsolvable models by GPT-4.

Comparing the two pipelines incorporated within this research, there was no statistically significant difference between the performance of the two approaches in mathematical modeling from natural language descriptions. Although the Chain-of-Thought prompting style was expected to improve the ability of GPT-4 in model formulation (Li et al., 2023), the research results do not show evidence for this. This may be due to the simplicity of prompts used in the Chain-of-Thought pipeline. A single sentence prompt was used, asking for "only" the required parts of the mathematical model at each prompting step. Although this encompasses the meaning of what is expected, previous research has shown that semantically similar but differently written prompts can lead to entirely different outcomes (Salinas and Morstatter, 2024). Prompt-engineering can be used to optimize these prompts, which may improve the performance of the Chain-of-Though pipeline.

In terms of mathematical modeling performance, the primary failure point of GPT-4 in this study was the formulation of constraints, a weakness also documented by Fan et al. (2024). These recurring failures may stem from the complexity of the constraint types in the dataset. However, the rest of the model components, like variables and objective function formulation, also fall short of expectations of an AI copilot. The low performance across all model components could be attributed to the limitations of the implemented pipelines. The pipelines in this research consist of a singular output achieved from the given prompts, without room for self correction of GPT-4. AhmadiTeshnizi et al. (2023) utilized a self-correction step in their methodology, allowing the LLM to correct its mistakes. Both pipeline 1 and pipeline 2 lack a self-correction stage, which could potentially be implemented to allow GPT-4 to identify and rectify its own errors in modeling. In addition, one-shot learning could be incorporated into the prompts to enhance modeling performance, as previously explored by Ahmed and Choudhury (2024). This approach involves

providing the language model with an example of a similar problem formulation, complete with correct parameters, variables, objective function and constraints, within the prompt. Such an example can serve as a guide, enabling GPT-4 to better comprehend the correct formulation of similar models and potentially lead to improved performance in model formulation.

Furthermore, the comparatively low mathematical model correctness scores may be linked to the binary evaluation metrics used to assess constraint correctness. These metrics assigns a score of 0 if even a single required component is incorrect or missing from the formulation. For instance, if GPT-4 correctly models five out of six constraints, the constraint score would still be 0. The same applies for the evaluation of parameters and variable formulations. While these binary metrics align with the research goal of evaluating model correctness for use by business professionals lacking expertise in OR, it does result in lower correctness scores. A more nuanced approach could involve calculating the mathematical model correctness metrics as a percentage of the required components. This may provide a more accurate reflection of GPT-4's performance in mathematical model formulation and help depict more specific areas of failure to be explored in further research.

Another notable finding of this research was the low consistency scores observed in the model outputs generated by GPT-4. In the context of an AI Copilot application, consistency in responses to business optimization problems is essential, ensuring that the optimal solution remains consistent across repeated questions. For business users, the ability to rely on consistent outputs is crucial for decision-making processes. One potential way to enhance the consistency of GPT-4 involves prompt-engineering techniques aimed at achieving greater uniformity in generated outputs. Prompt-engineering of LLMs in optimization has previously been experimented with, however, largely was focused on improving accuracy of outputs (Yang et al., 2023). However, in the field of medicine, Wang et al. (2024) explored how different prompts effect the consistency of the answers provided by the LLM. For future research, inspiration can be taken from this, and prompt-engineering specifically to improve consistency of outputs can be explored for optimization modeling with LLMs as well.

Furthermore, regarding the coding evaluations, translating models into executable code using GPT-4 demonstrates potential, yet still needs to be improved. Notably, GPT-4 shows promise by preserving the semantic meaning of the model during translation. However, challenges arise, particularly with the Pyomo syntax, resulting in compromised executability of the generated code. This issue could be mitigated by fine-tuning the model and by pre-informing GPT-4 with Pyomo syntax-related information, prior to problem prompts. Fine-tuning has been experimented with in previous optimization research (Ahmed and Choudhury, 2024), and is indicated to improve performance of LLMs in specific tasks (Wei et al., 2022).

Therefore, GPT-4 can be fine-tuned for optimization modeling and taught pyomo syntax in future research, to possibly improve performance.

Overall, the limitations of this study include the lack of domain-specific fine-tuning and prompt engineering to optimize the outputs received by the LLM, as suggested as improvements previously. Moreover, due to its scope, this research was conducted on only 16 problems. For further research, it is recommended to extend this dataset to get a more concrete idea of its ability to model problems across these problem types. In addition, the experimental pipelines currently involve no room for GPT-4 to clairfy and guide its reasoning process. An interactive methodology could be employed, where GPT-4 can ask questions and clarify its problem understanding with the business user. Such an approach would more accurately reflect the workflow of an AI Copilot, thereby providing a more realistic assessment of GPT-4's potential in practical applications.

6 Conclusion

Overall, the results of the experiments highlight that although GPT-4 shows some promise, there are still major gaps in the abilities of this state-of-the-art LLM in optimization modeling. Major improvements are required in mathematical modeling abilities. Significant failure patterns are observed across binary variable definitions, objective function formulations and constraint formulations. The consistency of outputs also requires refinements, before GPT-4 is leveraged for an AI Copilot application. Even though GPT-4 is able to model and produce solver code for optimization problems, entirely correct responses are still rare. Consequently, currently GPT-4 could be effectively utilized by experts in OR who possess the expertise to identify and rectify potential missing model parts and mistakes in overall formulations. This will definitely speed up the process of optimization modeling for OR professionals. Nevertheless, from the perspective of a general business user lacking OR knowledge, significant gaps remain in leveraging GPT-4 for more complex business optimization tasks.

7 References

Abdin, A. F., Fang, Y. P., Caunhye, A., Alem, D., Barros, A., and Zio, E. (2023). An Optimization Model for Planning Testing and Control Strategies to Limit the Spread of a Pandemic–The case of COVID-19. *European journal of operational research*, 304(1):308–324. 10.1016/j.ejor.2021.10.062.

AhmadiTeshnizi, A., Gao, W., and Udell, M. (2023). Optimus: Optimization modeling using mip solvers and large language models. https://doi.org/10.48550/arXiv.2310.06116.

Ahmed, T. and Choudhury, S. (2024). LM4OPT: Unveiling the Potential of Large Language Models in Formulating Mathematical Optimization Problems. https://doi.org/10.48550/arXiv.2403.01342.

Amarasinghe, P. T., Nguyen, S., Sun, Y., and Alahakoon, D. (2023). Ai-copilot for business optimisation: A framework and a case study in production scheduling. *arXiv preprint arXiv:2309.13218*. https://doi.org/10.48550/arXiv.2309.13218.

Anthropic (2024). The Claude 3 Model Family: Opus, Sonnet, Haiku. https://www-cdn.anthropic.com/de8ba9b01c9ab7cbabf5c33b80b7bbc618857627/Model_Card_Claude_3.pdf.

Bakas, N. P., Papadaki, M., Vagianou, E., Christou, I., and Chatzichristofis, S. A. (2024). Integrating LLMs in Higher Education, Through Interactive Problem Solving and Tutoring: Algorithmic Approach and Use Cases. In Papadaki, M., Themistocleous, M., Al Marri, K., and Al Zarouni, M., editors, *Information Systems. EMCIS 2023. Lecture Notes in Business Information Processing*, volume 501. Springer, Cham. 10.1007/978-3-031-56478-9_21.

Besta, M., Blach, N., Kubicek, A., Gerstenberger, R., Podstawski, M., Gianinazzi, L., Gajda, J., Lehmann, T., Niewiadomski, H., Nyczyk, P., and Hoefler, T. (2024a). Graph of thoughts: Solving elaborate problems with large language models. https://doi.org/10.48550/arXiv.2308.09687.

Besta, M., Memedi, F., Zhang, Z., Gerstenberger, R., Piao, G., Blach, N., Nyczyk, P., Copik, M., Kwaśniewski, G., Müller, J., Kubicek, A., Niewiadomski, H., O'Mahony, A., Mutlu, O., and Hoefler, T. (2024b). Demystifying Chains, Trees, and Graphs of Thoughts. *arXiv.org*. https://doi.org/10.48550/arXiv.2401.14295.

Birge, J. R. and Louveaux, F. (2011). *Introduction to Stochastic Programming*. Springer Science & Business Media.

Boiko, D. A., MacKnight, R., and Gomes, G. (2023). Emergent autonomous scientific research capabilities of large language models. https://doi.org/10.48550/arXiv.2304.05332.

Bracken, J. and McCormick, G. P. (1968). *Selected Applications of Nonlinear Programming*. https://apps.dtic.mil/sti/pdfs/AD0679037.pdf.

Bynum, M. L., Hackebeil, G. A., Hart, W. E., Laird, C. D., Nicholson, B. L., Siirola, J. D., Watson, J.-P., and Woodruff, D. L. (2021). *Pyomo—Optimization modeling in Python*. 3rd edition.

Castillo, E., Conejo, A. J., Pedregal, P., Garcia, R., and Alguacil, N. (2011). *Building and Solving Mathematical Programming Models in Engineering and Science*. John Wiley & Sons.

Chu, Z., Ni, S., Wang, Z., Feng, X., Li, C., Hu, X., Xu, R., Yang, M., and Zhang, W. (2024). History, Development, and Principles of Large Language Models—An Introductory Survey. https://arxiv.org/pdf/2402.06853.

Cornuejols, G. and Tütüncü, R. (2006). *Optimization Methods in Finance*, volume 5. Cambridge University Press.

De Angelis, L., Baglivo, F., Arzilli, G., Privitera, G. P., Ferragina, P., Tozzi, A. E., and Rizzo, C. (2023). ChatGPT and the Rise of Large Language Models: The New AI-driven Infodemic Threat in Public Health. 10.3389/fpubh.2023.1166120.

Dodge, J., Ilharco, G., Schwartz, R., Farhadi, A., Hajishirzi, H., and Smith, N. (2020). Fine-tuning Pretrained Language Models: Weight Initializations, Data Orders, and Early Stopping. *arXiv preprint arXiv:2002.06305*. https://doi.org/10.48550/arXiv.2002.06305.

Fan, Z., Ghaddar, B., Wang, X., Xing, L., Zhang, Y., and Zhou, Z. (2024). Artificial Intelligence for Operations Research: Revolutionizing the Operations Research Process. https://doi.org/10.48550/arXiv.2401.03244.

Gangwar, N. and Kani, N. (2023). Highlighting Named Entities in Input for Auto-Formulation of Optimization Problems. https://doi.org/10.48550/arXiv.2212.13201.

Gemini Team, G. (2024). Gemini 1.5: Unlocking Multimodal Understanding Across Millions of Tokens of Context. https://doi.org/10.48550/arXiv.2403.05530.

Google (2023). PaLM 2: Technical Report. Technical report, Google. Retrieved from https://ai.google/static/documents/palm2techreport.pdf.

Hart, W. E., Watson, J. P., and Woodruff, D. L. (2011). Pyomo: Modeling and Solving Mathematical Programs in Python. *Mathematical Programming Computation*, 3. https://doi.org/10.1007/s12532-011-0026-8.

He, J., N, M., Vignesh, S., Kumar, D., and Uppal, A. (2022). Linear Programming Word Problems Formulation Using EnsembleCRF NER labeler and T5 Text Generator with Data Augmentations. https://doi.org/10.48550/arXiv.2212.14657.

Jiang, A. Q., Sablayrolles, A., Roux, A., Mensch, A., Savary, B., Bamford, C., Chaplot, D. S., de las Casas, D., Bou Hanna, E., Bressand, F., Lengyel, G., Bour, G., Lample, G., Lavaud, L. R., Saulnier, L., Lachaux, M.-A., Stock, P., Subramanian, S., Yang, S., Antoniak, S., Le Scao, T., Gervet, T., Lavril, T., Wang, T., Lacroix, T., and El Sayed, W. (2024). Mixtral of Experts. https://doi.org/10.48550/arXiv.2401.04088.

Le, H., Wang, Y., Gotmare, A. D., Savarese, S., and Hoi, S. C. H. (2022). CodeRL: Mastering Code Generation through Pretrained Models and Deep Reinforcement Learning. https://doi.org/10.48550/arXiv.2207.01780.

Li, Q., Zhang, L., and Mak-Hau, V. (2023). Synthesizing Mixed-Integer linear Programming Models from Natural Language Descriptions. https://doi.org/10.48550/arXiv.2311.15271.

López Espejel, J., Ettifouri, E. H., Yahaya Alassan, M. S., Chouham, E. M., and Dahhane, W. (2023). GPT-3.5, GPT-4, or BARD? Evaluating LLMs Reasoning Ability in Zero-Shot Setting and Performance Boosting Through Prompts. https://doi.org/10.48550/arXiv.2305.12477.

Manyika, J. and Hsiao, S. (2023). An Overview of Bard: An Early Experiment with Generative AI. Retrieved from https://ai.google/static/documents/google-about-bard.pdf.

MOSEK (n.d.). 11.3 robust linear optimization — mosek optimization toolbox for matlab 10.2.0. Retrieved May 23, 2024, from https://docs.mosek.com/latest/toolbox/case-studies-robust-lo.html.

Naveed, H., Khan, A. U., Qiu, S., Saqib, M., Anwar, S., Usman, M., Akhtar, N., Barnes, N., and Mian, A. (2023). A Comprehensive Overview of Large Language Models. https://doi.org/10.48550/arXiv.2307.06435.

OpenAI (2022). Introducing chatgpt. Retrieved from https://openai.com/index/chatgpt/.

OpenAI (2023). Gpt-4. Retrieved from https://openai.com/index/gpt-4-research/.

OpenAI (2024a). GPT-4 technical report. Retrieved from https://doi.org/10.48550/arXiv.2303.08774.

OpenAI (2024b). Gpt-4o. Retrieved from https://openai.com/index/hello-gpt-4o/.

Poler, R., Mula, J., and Díaz-Madroñero, M. (2014). *Operations research problems*. Springer eBooks.

Radford, A. and Narasimhan, K. (2018). Improving language understanding by generative pre-training. Retrieved from https://api.semanticscholar.org/CorpusID:49313245.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. (2018). Language Models are Unsupervised Multi Task Learners. https://api.semanticscholar.org/CorpusID:160025533.

Raj, H., Rosati, D., and Majumdar, S. (2022). Measuring Reliability of Large Language Models Through Semantic Consistency. https://arxiv.org/pdf/2211.05853.pdf.

Ramamonjison, R., Yu, T. T., Li, R., Li, H., Carenini, G., Ghaddar, B., He, S., Mostajabdaveh, M., Banitalebi-Dehkordi, A., Zhou, Z., and Zhang, Y. (2023). NL4Opt Competition: Formulating Optimization Problems Based on Their Natural Language Descriptions. https://doi.org/10.48550/arXiv.2303.08233.

Salinas, A. and Morstatter, F. (2024). The Butterfly Effect of Altering Prompts: How Small Changes and Jailbreaks Affect Large Language Model Performance. *arXiv preprint arXiv:2401.03729*. https://doi.org/10.48550/arXiv.2401.03729.

Sarker, L., Downing, M., Desai, A., and Bultan, T. (2024). Syntactic Robustness for LLM-based Code Generation. Retrieved from https://arxiv.org/abs/2404.01535.

Stok, D. (2022). Optimization Truck Loading and Scheduling. Master's thesis, Erasmus University Rotterdam.

Toloka (2023). The History, Timeline, and Future of LLMs. Retrieved from https://toloka.ai/blog/history-of-llms/.

Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., Bikel, D., Blecher, L., Canton Ferrer, C., Chen, M., Cucurull, G., Esiobu, D., Fernandes, J., Fu, J., Fu, W., Fuller, B., and Edunov, S. (2023). Llama 2: Open Foundation and Fine-Tuned Chat Models. Retrieved from https://arxiv.org/pdf/2307.09288.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, , and Polosukhin, I. (2023). Attention is All You Need. arXiv:1706.03762. Retrieved from https://arxiv.org/abs/1706.03762.

Wang, L., Chen, X., Deng, X., Wen, H., You, M., Liu, W., Li, Q., and Li, J. (2024). Prompt engineering in consistency and reliability with the evidence-based guideline for llms. *npj Digital Medicine*, 7:41. https://doi.org/10.1038/s41746-024-01029-4.

Wasserkrug, S., Boussioux, L., Hertog, D. D., Mirzazadeh, F., Birbil, I., Kurtz, J., and Maragno, D. (2024). From Large Language Models and Optimization to Decision Optimization CoPilot: A Research Manifesto. https://doi.org/10.48550/arXiv.2402.16269.

Wei, J., Bosma, M., Zhao, V. Y., Guu, K., Yu, A. W., Lester, B., Du, N., Dai, A. M., and Le, Q. V. (2022). Finetuned Language Models Are Zero-Shot Learners. https://doi.org/10.48550/arXiv.2109.01652.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E. H., Le, Q. V., and Zhou, D. (2023). Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. https://doi.org/10.48550/arXiv.2201.11903.

Xiao, Z., Zhang, D., Wu, Y., Xu, L., Wang, Y. J., Han, X., Fu, X., Zhong, T., Zeng, J., Song, M., and Chen, G. (2024). Chain-of-Experts: When LLMs meet complex operations research problems. Retrieved from https://openreview.net/forum?id=HobyL1B9CZ.

Yang, C., Wang, X., Lu, Y., Liu, H., Le, Q. V., Zhou, D., and Chen, X. (2023). Large language models as optimizers. Retrieved from https://arxiv.org/abs/2309.03409.

Zhao, W. X., Zhou, K., Li, J., Tang, T., Wang, X., Hou, Y., Min, Y., Zhang, B., Zhang, J., Dong, Z., Du, Y., Yang, C., Chen, Y., Chen, Z., Jiang, J., Ren, R., Li, Y., Tang, X., Liu, Z., Liu, P., Nie, J.-Y., and Wen, J.-R. (2023). A Survey of Large Language Models. Retrieved from https://arxiv.org/pdf/2303.18223.

8 Appendices

Appendix 1

Problem	Name	Domain	Characteristics	Description
LP-1	Production Planning in a Cosmetics Firm	Production Planning	Capacity Constraints	Explicit
LP-2	Robust Drug Production	Production Planning	Capacity Constraints, Robust programming	Implicit
LP-3	Weekly Production in Metallurgical Company	Production Planning	Capacity Constraints	Explicit
LP-4	Farmers Problem	Production Planning	Varying Capacity Constraints	Explicit
IP-1	Radiotherapy Treatment	Assignment	Problem Specific Function	Implicit
IP-2	Choosing Investment Strategies	Portfolio Optimization	Mean-Variance modeling, Big-M and Chance Constraints	Implicit
IP-3	Bin Packing	Assignment	Logic Constraints	Implicit
IP-4	Managing Employee Working Hours	Scheduling	Logic Constraints, Positive or Negative deviation in variables	Explicit
MIP-1	Facility Location Allocation	Location Allocation	Logic Constraints	Implicit
MIP-2	Food and Beverage Production	Production Planning	Time-steps, Different Unit representations	Both
MIP-3	Thermal Power Plant Scheduling	Production Planning	Time-steps, Logic Constraints Robust programming	Both
MIP-4	Water Flow Network Planning	Network Problem	Time-steps, Logic Constraints	Explicit
NLP-1	Production Planning in a Drinks Firm	Production Planning	Different unit representations	Explicit
NLP-2	Planning Materials in Chemistry	Production Planning	Logarithmic Objective Function	Explicit
NLP-3	Bid Evaluation	Selection	Logic Constraints, Piece-wise function	Explicit
NLP-4	Alkylation Process	Process Optimization	Different unit representations, Robust programming	Both

Table A1: Data Overview

The problem types in this research are categorized as Linear Programming (LP), Integer Programming (IP), Mixed Integer Programming (MIP), and Nonlinear Programming (NLP), and numbered 1 through 4.

Table A1 shows an overview of the 16 problems utilized in the experiments. For each problem, the table details its name, domain, main characteristics, and categorizes its worded problem description as implicit, explicit, or both.

Appendix 2

LP-1 Problem Description:

"A firm from Milan sells chemical products for professional cosmetics. It is planning the production of three products, GCA, GCB and GCC, for a given period of time by mixing two different components: C1 and C2. All the end products must contain at least one of the two components, and not necessarily both. For the next planning period, 10,000 l of C1 and 15,000 l of C2 are available. The production of GCA, GCB and GCC must be scheduled to at least cover the minimum demand level of 6,000, 7,000 and 9,000 l, respectively. It is assumed that when chemical components are mixed, there is no loss or gain in volume. Each chemical component, C1 and C2, has a proportional critical element, 0.4 and 0.2, respectively. That is to say, each litre of C1 contains 0.4 l of the critical element. To obtain GCA, the mixture must proportionally contain at least a 0.3 fraction of the critical element. Another requirement is that the quantity of the critical element is seen in GCB, an 0.3 fraction at the most. Furthermore, the minimum ratio of C1 with C2 in product GCC must be 0.3. The profit expected for the sale of each litre of GCA, GCB and GCC is $120, $135 and $155, respectively. Optimise the production planning of this firm."

IP-1 Problem Description:

"We are delighted to welcome you, our newest intern on the Analytics team of Massachusetts General Hospital! You have been placed in a challenging role where you will be tasked with solving a real-world problem in the field of medical physics. We are building a pilot program in Boston, and if successful, your work could be applied widely in hospitals with limited capacity in many countries. You are responsible for determining the best treatment plan for 17 patients who require radiotherapy. Your goal is to optimize the use of two possible treatments: photon therapy and proton therapy. While proton therapy is known to target tumors more precisely, it is also more expensive and has limited capacity in many countries. Therefore, you will need to balance the benefits of proton therapy with its limitations and cost to create an effective treatment plan for each patient. To determine the best course of action for each patient, you will use a scoring system called the Biological Equivalent Dose (BED). This system allows you to calculate the effectiveness of each patient's treatment plan by considering the number of proton fractions that can be used

while still achieving the highest possible BED.

We have n=17 patients who need radiotherapy. Each patient i needs 15 fractions, which can be photon fractions, proton fractions, or a mix of photon and proton fractions (e.g. 4 proton fractions and 11 photon fractions). We want to use the limited proton therapy capacity as best as possible. We can calculate the BED score for each patient when p proton fractions and 15-p photon fractions are used, as BEDi(p,15-p), i.e., the BED when p proton and 15-p photon fractions are delivered for patient i. The higher the score, the better. The data file "ProblemData.csv" contains a 2D matrix of BED scores. It does not have an index. It was made in Excel and saved as csv. The columns are the number of proton fractions and each row represents a patient. In particular, the number at the (i,j) position is the score for patient i receiving j proton fractions. Suppose that the total maximal capacity C is 100 proton fractions. To maximize the total BED scores for all the patients, which patients should get proton fractions, and how many should they get? Formulate an integer linear optimization model to solve this problem. Assume you know the value BEDi(j,15-j) for each patient i."

MIP-1 Problem Description:

"You are a city planner, looking to open facilities at some locations. We have a set of customers and a set of possible locations for opening facilities. Each potential location for establishing a facility incurs a fixed annual activation cost, which must be paid if the facility is used, regardless of the service volume it handles. Furthermore, this service volume at each facility is capped at a maximum annual limit. Additionally, there are transportation costs associated with servicing each customer from each facility.

The goal is to minimize the overall costs, which include both the fixed activation costs for any opened facilities and the transportation costs for servicing customers. This must be done while making sure that each customer's demand is met, each facility does not exceed its maximum service volume, and customers can of course only be serviced by a facility that is opened. Please formulate this as a mathematical optimization model."

NLP-1 Problem Description:

"A firm that packs refreshments and beers, situated in the province of Valencia (Spain) employs the same syrup to produce its 1.5 l COLI and PEPSA products on its S1 production line. Once processed, each hectolitre of syrup produces 40 units of the 1.5 l COLI product and 20 units of the 1.5 l PEPSA product. If X1 units of the 1.5 l COLI product and X2 units of the 1.5 l PEPSA product are produced, the firm estimates that the daily income obtained in dollars would be given by the following function:

49000 times X1 minus X1 squared plus 30 times X2 minus two times X2 squared. It costs 150 dollars to buy and process each hectolitre of syrup. The S1 packaging line has a net capacity of producing 7100 1.5 l product units every hour. The firm works 5 days a week in 8h shifts. Given its weekly target coverage, the firm is committed to produce at least half the amount of PEPSA than COLI. Although priority orders tend to amend its production planning, the firm wishes to have a basic product planning that optimizes its daily profits."

Appendix 3

Type	Parameter Correctness	Variable Correctness	Objective Function Correctness	Constraint Correctness	Mathematical Model Correctness	Model Solution Optimality
LP-1	1.0	0.67	1.0	0.0	0.0	0.67
LP-2	0.0	0.33	0.0	0.0	0.0	0.0
LP-3	0.67	1.0	0.33	0.33	0.0	0.0
LP-4	0.67	0.33	0.0	0.0	0.0	0.0
LP	**0.58**	**0.58**	**0.33**	**0.08**	**0.0**	**0.17**
IP-1	1.0	0.33	0.67	0.33	0.33	0.33
IP-2	1.0	0.33	0.67	0.67	0.33	0.33
IP-3	1.0	0.67	0.67	0.0	0.0	0.33
IP-4	0.67	0.0	0.0	0.0	0.0	0.0
IP	**0.92**	**0.33**	**0.50**	**0.25**	**0.17**	**0.25**
MIP-1	1.0	1.0	1.0	0.0	0.0	0.67
MIP-2	0.67	0.0	0.0	0.0	0.0	0.0
MIP-3	0.67	0.67	0.67	0.0	0.0	0.0
MIP-4	1.0	0.0	0.0	0.0	0.0	0.0
MIP	**0.75**	**0.42**	**0.42**	**0.0**	**0.0**	**0.17**
NLP-1	1.0	1.0	1.0	0.0	0.0	0.67
NLP-2	1.0	1.0	1.0	0.33	0.33	0.33
NLP-3	1.0	0.0	0.0	0.0	0.0	0.0
NLP-4	1.0	0.67	1.0	0.0	0.0	0.0
NLP	**1.0**	**0.67**	**0.75**	**0.08**	**0.08**	**0.25**
ALL	**0.83**	**0.50**	**0.48**	**0.10**	**0.06**	**0.21**

Table A2: Pipeline 1 Results for MM1, MM2 anf MM3

Type	Parameter Correctness	Variable Correctness	Objective Function Correctness	Constraint Correctness	Mathematical Model Correctness	Model Solution Optimality
LP-1	1.0	1.0	1.0	0.33	0.33	0.67
LP-2	0.0	1.0	1.0	0.0	0.0	0.0
LP-3	1.0	1.0	0.33	0.67	0.33	0.33
LP-4	1.0	0.67	1.0	0.0	0.0	0.0
LP	**0.75**	**0.92**	**0.83**	**0.25**	**0.17**	**0.25**
IP-1	1.0	0.0	0.33	0.0	0.0	0.0
IP-2	1.0	0.33	0.0	0.0	0.0	0.0
IP-3	1.0	0.67	0.67	0.0	0.0	0.67
IP-4	1.0	0.0	0.0	0.0	0.0	0.0
IP	**1.0**	**0.25**	**0.25**	**0.0**	**0.0**	**0.17**
MIP-1	1.0	0.33	0.33	0.67	0.33	0.33
MIP-2	1.0	0.0	0.0	0.0	0.0	0.0
MIP-3	0.33	0.33	0.33	0.67	0.33	0.33
MIP-4	1.0	0.33	0.33	0.0	0.0	0.0
MIP	**0.83**	**0.25**	**0.25**	**0.17**	**0.08**	**0.08**
NLP-1	1.0	1.0	1.0	0.0	0.0	1.0
NLP-2	1.0	1.0	1.0	0.33	0.33	0.33
NLP-3	0.0	1.0	0.33	0.67	0.0	0.0
NLP-4	0.33	1.0	0.33	0.67	0.0	0.0
NLP	**0.83**	**0.75**	**0.58**	**0.25**	**0.08**	**0.33**
ALL	**0.84**	**0.54**	**0.48**	**0.17**	**0.08**	**0.21**

Table A3: Pipeline 2 Results for MM1, MM2 anf MM3

Type	Code Equivalence to Model	Code Executability	Code Solution Optimality
LP-1	0.67	0.33	0.33
LP-2	1.00	0.67	0.00
LP-3	0.33	0.67	0.00
LP-4	0.00	0.33	0.00
LP	**0.50**	**0.50**	**0.08**
IP-1	0.67	1.00	0.33
IP-2	1.00	0.67	0.33
IP-3	1.00	0.67	0.00
IP-4	0.67	0.33	0.00
IP	**0.83**	**0.67**	**0.17**
MIP-1	0.67	1.00	0.33
MIP-2	0.67	0.33	0.00
MIP-3	1.00	0.33	0.00
MIP-4	0.33	0.67	0.00
MIP	**0.67**	**0.58**	**0.08**
NLP-1	1.00	1.00	0.67
NLP-2	0.67	0.00	0.33
NLP-3	0.00	0.33	0.00
NLP-4	0.00	0.67	0.00
NLP	**0.42**	**0.50**	**0.25**
ALL	**0.60**	**0.56**	**0.15**

Table A4: Pipeline 1 Results for MC1, MC2 and MC3

Type	Code Equivalence to Model	Code Executability	Code Solution Optimality
LP-1	1.0	0.67	0.67
LP-2	0.67	0.33	0.0
LP-3	1.0	1.0	0.33
LP-4	0.33	0.33	0.0
LP	**0.75**	**0.58**	**0.25**
IP-1	1.0	0.67	0.0
IP-2	0.0	0.67	0.0
IP-3	1.0	1.0	0.33
IP-4	0.33	0.67	0.0
IP	**0.58**	**0.75**	**0.08**
MIP-1	0.67	1.0	0.0
MIP-2	1.0	0.33	0.0
MIP-3	0.33	0.0	0.0
MIP-4	0.0	0.33	0.0
MIP	**0.50**	**0.42**	**0.0**
NLP-1	0.67	1.0	1.0
NLP-2	1.0	0.0	0.33
NLP-3	0.33	0.33	0.0
NLP-4	0.0	0.33	0.0
NLP	**0.50**	**0.42**	**0.33**
ALL	**0.58**	**0.54**	**0.17**

Table A5: Pipeline 2 Results for MC1, MC2 and MC3

Appendix 4

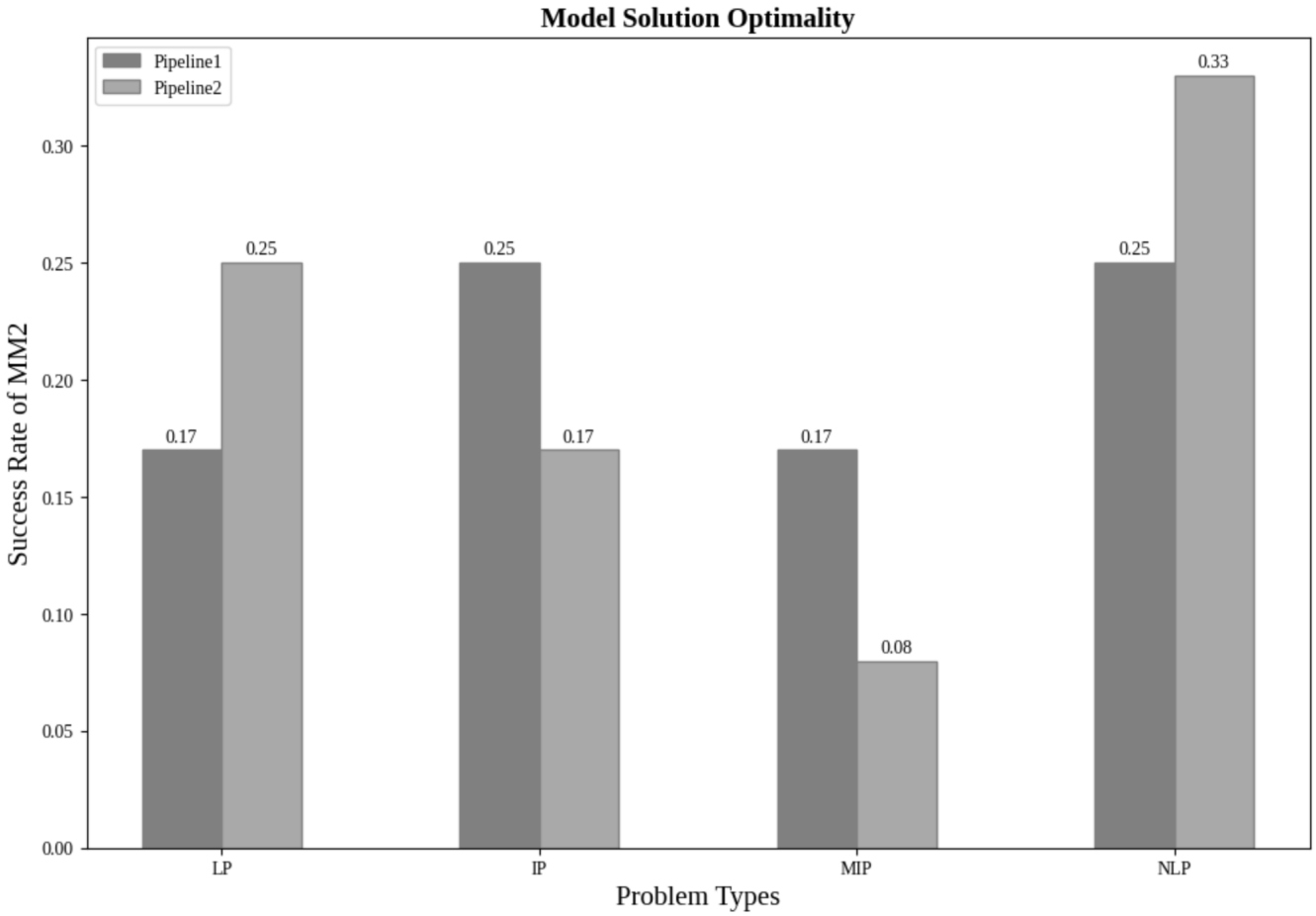


Figure 7: This figure shows the Model Solution Optimality (MM2) scores across each problem type.