Universiteit van Amsterdam
Amsterdam School of Economics

Claude 3 Opus for Automatic Formulation and Solution of Optimization Models

Annika Siefke

Bachelor's Thesis to obtain a degree in
Business Analytics
University of Amsterdam
Faculty of Economics and Business
Amsterdam School of Economics

Author: Annika Siefke
Student nr: 13152505
Date: June 26, 2024
Supervisor: Donato Maragno

Abstract

This thesis explores the potential of using the Large Language Model (LLM) Claude 3 Opus to automate the process of formulating and solving mathematical optimization problems. By leveraging Claude 3 Opus, this research aims to bridge the expertise gap in organizations lacking specialized Operations Research (OR) professionals. Using a one-step prompting approach as well as a multi-step prompting approach, this study involves a series of experiments evaluating Claude 3 Opus's ability to translate business problems from natural language descriptions into mathematical models and executable code. The dataset contains 16 problems spanning various optimization classes (Linear Programming (LP), Mixed-Integer Programming (MIP), Integer Programming (IP), and Nonlinear Programming (NLP)) and complexity levels. Key performance metrics include the correctness of the mathematical model, solution optimality, consistency, and code executability. *Claude 3 Opus* demonstrated a fairly consistent ability to extract parameters, decision variables, and objective functions but was statistically significantly worse at modeling constraints, especially with complex constraint types such as logic, robust, or time-dynamic constraints, and when prompts were not explicit enough. Translating mathematical models into executable code was generally successful, with most errors arising from specific syntax issues in Pyomo. Overall, *Claude 3 Opus* solved 41.67% of problems using the one-step prompting approach and 45.83% of problems using the multi-step prompting approach. This difference between the two approaches was not statistically significant. The research suggests that while *Claude 3 Opus* has potential, it cannot yet serve as an autonomous Optimization CoPilot. Further improvements and research are needed to enhance its reliability and performance in practical applications.

Keywords Large Language Model, Claude 3 Opus, Automatic Formulation, Mathematical Optimization, Linear Programming, Integer Programming, Mixed Integer Programming, Nonlinear Programming, Pyomo

Statement of Work

This thesis includes both individual and collaborative elements. Chapter 3 Methodology is predominantly based on group work. All the other sections in this thesis are my individual contributions. I, Annika Siefke, take full responsibility for all work presented in this thesis irrespective of it being based on group or individual work. 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. 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.

Contents

1 Introduction

Many industries face complex decision-making problems, such as allocating testing and control resources during the COVID-19 pandemic, optimizing conjunctive use of available water resources for irrigation, or finding ideal parameters in a cryptocurrency trading algorithm (Abdin et al., 2023; Singh, 2012; Omran et al., 2023). To make the best decision and reap financial or societal benefits, organizations can apply mathematical optimization to determine the optimal solution to the given problem. Commonly, experts of the *Operations Research* (OR) industry find solutions to such issues as most decision-makers lack the expertise to apply required mathematical techniques themselves. After translating optimization problems into mathematical formulas consisting of decision variables, an objective function, and a set of constraints, OR experts can then use state-of-the-art solvers such as *Gurobi* that implement various advanced optimization algorithms (Gurobi Optimization, LLC, n.d.) to solve the problem at hand. Doing so will identify an optimal solution to the problem given the constraints enforced by the decision-making context, which can then be used to optimize the business process. This can not only result in significant cost savings but can have other positive social effects as well, depending on the problem. However, due to dependence on OR experts, many organizations do not make use of mathematical optimization in their business processes.

Large Language Models (LLMs) such as *Claude 3 Opus* (Anthropic, 2024) offer a potential way to bridge this expertise gap. With these tools, everyone now has access to state-of-the-art *artificial intelligence* (AI) technologies that can assist the user in a wide variety of tasks like data analysis (OpenAI, 2023), assisting in academic research (Sider, 2023) or efficiently writing code (Prompt Shell Smith, 2023). Namely, building an Optimization CoPilot using an LLM like *Claude 3 Opus* to automate the process of formulating and solving mathematical optimization problems from natural-language descriptions would enable decision-makers themselves to solve optimization problems within their organizations.

This thesis aims to investigate the viability of such an Optimization CoPilot. By performing a series of experiments aimed at studying the capabilities of the LLM *Claude 3 Opus* at formulating and solving mathematical optimization problems, this research hopes to answer the question **"How well are cutting-edge LLMs suited for the task of formulating and solving mathematical optimization models of varying problem classes and complexity levels form natural-language descriptions?"**. Specifically, the experiments aim to explore several of *Claude 3 Opus* qualities related

to optimization modeling: the ability to synthesize a correct mathematical model from a natural language description, to translate this model into executable code, and to generate consistent answers. The outcomes of this thesis will contribute to the available literature in the following ways:

1. **Novel Dataset**: A new dataset consisting of a wide range of real-world optimization problems is introduced. The dataset will be publicly available on GitHub[1] and can serve as a basis for future research in the field of LLMs for optimization.

2. **Novel LLM**: *Claude 3 Opus*, released in March 2024, represents a state-of-the-art LLM that has not previously been studied in OR.

3. **Novel Model Evaluation**: The experiments build on previous research but focus on exploring a wider range of the LLMs' capabilities. The research will not only evaluate to what extent *Claude 3 Opus* can arrive at a correct solution but also how consistently it does so and what factors contribute to that. Doing so will provide insights into differences in skills of *Claude 3 Opus* at solving optimization problems of various classes and complexity levels and what factors contribute to such differences.

This research is part of a wider University project conducted in collaboration with Melis Doga Cevik, Nathan Ferchtandiker, and Hubert Perliński. Each project group member performed the research described in chapter 3 Methodology using their chosen LLM. The employed LLMs were *GPT-4* (Achiam et al., 2023), *Mixtral 8x22b* (Mistral AI, 2024) and *Gemini 1.5 Pro* (Reid et al., 2024), respectively. The results section of this work will solely focus on the outcomes of experiments conducted using *Claude 3 Opus*. A graphical comparison between the different LLMs can be found in the Combined Results section in the Appendix and the shared GitHub[1] repository.

[1] https://github.com/RiVuss/LLMsForOptimizationModelling

2 Literature Review

As mentioned in chapter 1 Introduction, modern, state-of-the-art LLMs offer a promising way to democratize the ability to solve optimization problems across various industries. This would be especially beneficial for smaller companies who often lack the funds to hire OR experts and, hence, are forced to forgo benefits provided by using modern methods to optimize business processes. This chapter outlines the academic foundations of core concepts discussed in this thesis, namely mathematical optimization, large language models focusing on the chosen LLM, *Claude 3 Opus*, and their applications in optimization modeling.

2.1 Mathematical Optimization

Optimization represents a mathematical process in which an *objective function* consisting of several *decision variables* is maximized or minimized while usually meeting a set of requirements referred to as *constraints* (Cornuejols & Tütüncü, 2006). As the name suggests, the *objective function* represents a company's goal, such as minimizing costs or maximizing profit. The *decision variables* are factors the business can influence, such as the amount of product produced or the number of units bought from a retailer. Lastly, *constraints* mathematically constitute the contextual requirements and limitations like a maximum production capacity or a limited budget for purchasing goods. Optimization problems are modeled as below:

$$\begin{aligned} &\text{Minimize } f(x) \\ &\text{Subject to } g(x) \leq b \\ &\qquad\qquad h(x) = d \end{aligned}$$

While the above model shows a generic mathematical formulation, many classes of optimization problems exist. *Linear Programming* (LP) refers to a group of decision problems where the *objective function* as well as the *constraints* are linear functions and where the *decision variables* are elements of $\mathbb{R}^n$ (Cornuéjols et al., 2018a). Next to that, there is *Mixed Integer Programming* (MIP) and *Integer Programming* (IP). Both optimization classes differ from LP in the domain of the *decision variables*: where in LP $x \in \mathbb{R}^n$, some or all $x \in \mathbb{Z}^n$ in MIP and IP, respectively (Cornuéjols et al., 2018b; Cornuejols & Tütüncü, 2006). Finally, *Nonlinear Programming* (NLP) refers to the

process of optimizing decision problems that include nonlinear terms, which can either be part of the *objective function*, *constraints* or both (Cornuéjols et al., 2018c).

The exact technical methods for solving such optimization problems fall outside the scope of this thesis. However, there are many state-of-the-art solvers such as *Gurobi* that implement algorithms like the *Simplex Method* for solving LPs, the *Branch and Bound Method* for solving MIPs and IPs or *Generalized Reduced Gradient* for solving NLPs (Dantzig, 1990; Land & Doig, 1960; Lasdon et al., 1978).

2.2 Large Language Models

The term *Large Language Models* refers to a group of AI algorithms designed to process and produce human-like speech. This is achieved by training some form of deep learning models, such as recurrent neural networks, on large amounts of human-generated texts (Vaswani et al., 2017; Devlin et al., 2018). The resulting model has learned statistical patterns and probabilities from the training data, enabling it to emulate human speech (Radford et al., 2019). The term *large* in LLM refers to the number of parameters the resulting model will have, ranging from millions to even trillions. For most proprietary LLMs, the exact number of parameters is not revealed. Anthropic, for example, does not disclose the size of their newest family of models, Claude 3. However, scholars estimate that *Claude 3 Opus* has a size of around 2 trillion parameters (Thompson, 2024; Wadhwani, 2024).

The reason for using *Claude 3 Opus* for the research is twofold. *Claude 3 Opus* has been released in March 2024. Not only does it represent the current state-of-the-art of LLMs, but it also has not yet been studied in mathematical optimization due to its recency. Hence, using it for the experiments conducted as part of this thesis fulfills an exploratory purpose. The second reason for choosing *Claude 3 Opus* compared to other novel LLMs is its reported logical reasoning and code generation ability. Anthropic (2024) report high scores in several mathematical and programmatic benchmarks for *Claude 3 Opus* compared to GPT-4, GPT-3.5, Gemini 1.0 Ultra, Gemini 1.5 Pro and Gemini 1.0 Pro. For example, in a four-shot test using the MATH dataset (Hendrycks et al., 2021), *Claude 3 Opus* achieved a success rate of 61%, which is 2.5% higher than the second best score achieved by Gemini 1.5 Pro. In addition, Anthropic (2024) report that *Claude 3 Opus* scores the highest on the HumanEval dataset (Chen et al., 2021), which assesses the ability of LLMs to complete Python programming tasks. With a score of 84.9%, *Claude 3 Opus* outperforms Gemini 1.0 Ultra, the second highest scorer, by 11.5% on the zero-shot test. Both of these abilities are imperative for synthesizing a mathematical model from a decision-making problem and translating it into executable code.

2.3 Large Language Models as Optimizers

Since the release of the first LLMs, much research has been conducted investigating their abilities in various fields. Initial research has focused on more fundamental skills of the LLMs, such as solving mathematical equations (Cobbe et al., 2021; Hendrycks et al., 2021) or common sense reasoning (Sakaguchi et al., 2021; Zellers et al., 2019). With advancing training techniques and model sizes, the mathematical or common sense reasoning test performance has improved. For example, one of Anthropic's initial LLMs, *Claude Instant 1.1* showcased a success rate of 80.9% on the *GSM8K* assessment (Cobbe et al., 2021), which measures the model's ability to complete mathematical tasks on grade school level (Anthropic, 2023). Subsequent releases of LLMs in the *Claude* family, *Claude 2* and *Claude 3 Opus*, boasted increased scores of 88% and 95%, respectively (Anthropic, 2023; Anthropic, 2024). Such advancements in the models' fundamental skills motivated researchers to further investigate what state-of-the-art LLMs are capable of by applying them to more advanced tasks.

Early research on using LLMs in OR was conducted by Ramamonjison, Yu, Li, et al. (2023). One goal of their NL4OPT competition was to explore using LLMs for an automated approach to formulating mathematical models from a natural-language description of linear programming problems. As part of the competition, various teams worked on designing a method to achieve this goal using different language models such as *BART* (Lewis et al., 2019) and *T5* (Raffel et al., 2020) (Gangwar & Kani, 2023; Jang, 2022; Ning et al., 2023). Ramamonjison, Yu, Li, et al. (2023) assessed the accuracy of the mathematical model formulated by these language models. Gangwar & Kani (2023) achieved the highest accuracy score, 0.899, using an input tagging method combined with asking *BART* to generate the whole mathematical model at once. The input tagging method employed by Gangwar & Kani (2023) consisted of surrounding relevant text in the problem description with their optimization entities. For example, in the sentence below, the term *"at least"* would be surrounded by the XML-like tags `<CONST_DIR>` and `</CONST_DIR>`.

A farmer must produce at least two tons of wheat and corn to feed his animals.

The other top-scoring teams used similar methods of data augmentation via entity embedding or input tagging (Jang, 2022; Ning et al., 2023; Yang & Wang as cited in Ramamonjison, Yu, Li, et al. (2023); He et al., 2022). However, not all teams asked the model to generate the mathematical formulation in one go. He et al. (2022), for example, opted for a multi-task approach in which the chosen language model, *T5*, separately generated each equation of the model. Interestingly, while still yielding the fifth-best accuracy score in the NL4OPT competition, with an accuracy of 0.780, this approach is 8.6 percentage points behind the next top scorer. Subsequently, Ramamonjison, Yu, Li, et al. (2023) used *GPT-3.5-turbo* and a simple input-output setup without modifying the problem description. Using this model - a significantly larger model than both *BART*

and *T5* - yielded an accuracy score of 0.927, 2.8 percentage points better than the top scorer of the NL4OPT competition. Ahmed & Choudhury (2024) also use the NL4OPT dataset to test a fine-tuned version of *Llamba-2-7b*, *GPT-3.5* and *GPT-4* using their zero- and one-shot methodology. The two setups differ in that the one-shot method provides an example optimization problem and the corresponding mathematical model in the input prompt. In contrast, the zero-shot prompt only provides the mathematical model without a corresponding problem description. Ahmed & Choudhury (2024) report the highest F1 score of 0.6330 for the one-shot approach using *GPT-4*. However, both groups of researchers, Ramamonjison, Yu, Li, et al. (2023) and Ahmed & Choudhury (2024), also cautioned that the problems used in their studies might lack complexity and noted the need for future research to investigate whether this level of performance extends to more realistic use cases. Another limitation of the methodology as pointed out by Ramamonjison, Yu, Li, et al. (2023) is that it does not test the reliability of the LLMs' answers when tasked with formulating the same problem multiple times. Finally, neither Ramamonjison, Yu, Li, et al. (2023) nor Ahmed & Choudhury (2024) have investigated the abilities of LLMs to solve the mathematical models they generate by extending the research with a code generation and execution step.

Several researchers have subsequently addressed the first and third limitations of the research mentioned above. AhmadiTeshnizi et al. (2023) propose OptiMUS, an agent for synthesizing and solving optimization problems. They use *GPT-3.5* and *GPT-4* models for their experiments on a dataset consisting of LP and MIP decision-making problems. The models are tasked to first generate a mathematical model from a *Structured Natural Language Optimization Problem* (SNOP), which contains the optimization problem itself alongside some additional characteristics such as problem type, the input format of data files, or the solver to use. Subsequently, the model has to generate the code that corresponds to the mathematical formulation in a step-by-step approach, starting with the *decision variables*, followed by the *constraints*, and finally, the *objective function*. Additionally, AhmadiTeshnizi et al. (2023) introduce an additional step in their methodology, which automatically corrects or tests the generated solution. In the *Debug* step, for example, the process returns the error message alongside the faulty code and the mathematical model to the LLM and prompts it to fix the script. For *GPT-4*, adding this *Debug* step to the pipeline improved the success rate from below 0.40 in the baseline setup to around 0.48. Notably, AhmadiTeshnizi et al. (2023) report the opposite for *GPT-3.5*: success rate decreased to around 0.12 when introducing the *Debug* step as compared to 0.20 for the baseline.

Li et al. (2023) propose a three-stage method for designing mathematical models from unstructured natural language problem description. Initially, the model must identify the *decision variables*. In the next step, a fine-tuned version of the LLM classifies the constraints mentioned in the natural language input into different categories, such as *"Upper bound on single variable"* or *"If A then B/ if not B then not A/ B if*

A". As a final step, the same LLM used for step 1 needs to generate the constraints of mathematical form based on their natural language descriptions augmented with the constraint classes. Li et al. (2023) test their methodology using *GPT-3.5* and *PaLM 2* on a set of MIP problems. The results are compared to a zero-shot prompting approach using the respective LLM's chatbots. Using the multi-stage approach resulted in significantly higher accuracy scores. Li et al. (2023) report a model generation accuracy of 0.8667 and 0.7 for *GPT-3.5* and *PaLM 2* using their methodology. In the baseline condition, the *GPT-3.5* based chatbot achieved an accuracy of 0.2667 while *PaLM2's* chatbot scored 0.1667.

Amarasinghe et al. (2023) study the application of LLMs to auto-formulating mathematical optimization problems in the domain of *Job Shop Scheduling* (JSS), a type of IP. The proposed approach entails tasking a fine-tuned LLM to generate a code to solve the provided JSS problem. For their research, Amarasinghe et al. (2023) fine-tune *CodeRL* (Le et al., 2022). The fine-tuned LLM is then tasked across multiple steps to generate an optimization model as code. In total, the process of code generation was split into nine sub-tasks like *DEFINE_IMPORTS* or *DEFINE_OBJECTIVE*. The researchers analyze the impact of certain fine-tuning parameters on the success rate, defined as the proportion of correct outputs across all problems. Amarasinghe et al. (2023) note the positive effect of the number of epochs used in the fine-tuning on the success rate of their model, achieving near-perfect success rates when they used eight epochs.

Another multi-stage approach to automatically synthesizing optimization models using *GPT-4-vision-preview* was proposed by Huang et al. (2024). The researchers designed a three-step methodology that tasks the LLM to solve *Capacitated Vehicle Routing Problems* (CVRP). The first step, labeled *heuristic extraction*, provides an example CVRP problem and its solutions and asks the LLM to capture heuristics learned from the provided solutions. In the second step, a novel CVRP problem is provided, which the LLM must solve by adhering to the previously learned heuristics. The final step of the proposed method is the validation step. The solution provided by the LLM is evaluated for correctness. In case of any errors, these are returned to the LLM, and a valid solution is reached via multiple iterations. In contrast to other researchers, Huang et al. (2024) did not solely provide natural-language problem descriptions but used multi-modal input. Due to the nature of CVRP problems, Huang et al. (2024) hypothesized that adding a graphical representation of the described problem could aid the LLM in reaching a valid solution. The reported results support this theory as in 15 out of the 17 cases the multi-modal input format outperformed a purely text-based input format.

Finally, Xiao et al. (2024) offer a more advanced design to study the automation of formulating and solving optimization problems using LLMs. The proposed *"Chain-of-Experts"* (CoE) solution consists of a collection of *GPT-3.5-turbo* based agents referred to as *experts*. Each expert handles a specific sub-task of synthesizing and solving a natural-language-based decision-making problem. Some experts execute the technical tasks, like the *Modelling Expert* responsible for formulating the mathematical model and

the *Programmer* responsible for translating this model into code. Other experts, such as the *Evaluator*, are responsible for process-related tasks like verifying the code generated by the *Programmer* executes correctly. This setup is then tested on the dataset used in the NL4OPT competition in addition to the researchers' own *Complex OR* dataset, which reportedly more closely reflects the complexity of real-world optimization problems. To assess the performance of the proposed CoE, Xiao et al. (2024) use an accuracy measure that reflects whether or not the code generated by the CoE successfully passes the predefined test cases. Using the CoE methodology, Xiao et al. (2024) achieved an accuracy of 58.9% on the NL4OPT dataset and a score of 25.9% on the *Complex OR* dataset. The researchers compare the results to various other approaches, including the method used by Gangwar & Kani (2023) in the NL4OPT competition, resulting in accuracy scores of 47.9% and 0% on the two datasets. Xiao et al. (2024) also performed further experiments highlighting the importance of the self-correction mechanism their setup allows. Without the ability to reflect on the generated answers, the CoE setup results in a slightly lower accuracy of 22.7% on the *Complex OR* problems.

Previous research in the domain of using LLMs for automatically synthesizing and solving optimization problems has investigated two main prompting methods: *all-at-one* model generation (Jang, 2022; Ning et al., 2023; Yang & Wang as cited in Ramamonjison, Yu, Li, et al. (2023); Ahmed & Choudhury, 2024; mathematical modeling task in AhmadiTeshnizi et al., 2023; Huang et al., 2024; Xiao et al., 2024) and *step-by-step* model generation (He et al., 2022; code generation task in AhmadiTeshnizi et al., 2023; Li et al., 2023; Amarasinghe et al., 2023). However, no direct comparison between the two approaches was drawn. This research will directly compare these two methods to investigate whether either approach helps the LLM arrive at a correct solution that can have direct implications for developing an LLM-based Optimization CoPilot. Previous research has also mainly used OpenAI's GPT models to study the capabilities of LLMs related to optimization modeling (Ahmed & Choudhury, 2024; AhmadiTeshnizi et al., 2023; Huang et al., 2024; Ramamonjison, Yu, Li, et al., 2023; Li et al., 2023; Xiao et al., 2024). This research studies Anthropic's novel *Claude 3 Opus* which has not yet been studied in the context of operations research. *Claude 3 Opus'* capabilities in logical reasoning and code generation make it a promising candidate for use as an AI-based optimization assistant. The results of this research can be used to determine whether such capabilities might have potential advantages for the task at hand by drawing a comparison between the results reported by all project group members. Finally, another point that has largely been left unaddressed in the presented studies is the consistency in the answers generated by the LLM. As LLMs are probabilistic models, the generated outcomes are not deterministic and might vary from query to query. This is another point the research presented in this thesis will investigate.

3 Methodology

As described in chapter 2 Literature Review, previous studies have taken various approaches to studying the use of LLMs in OR. Most studies have focused on designing a methodology that yields the best possible outcomes given a particular set of problems rather than highlighting the strengths and weaknesses of LLMs concerning optimization modeling. In contrast, the research presented in this thesis aims to provide insights into the general abilities and failures of *Claude 3 Opus* at modeling a wide range of optimization problems. The results can be valuable for designing a real Optimization CoPilot as they point out specific failure patterns of the LLM that need to be addressed when developing such a tool. To this end, the research group collaboratively created the dataset, designed the pipelines, and developed the metrics presented in the section below.

3.1 Data Description

The dataset used to conduct the experiments has been curated with the goal of assessing *Claude 3 Opus'* capabilities of formulating and solving a wide range of real-world optimization problems. One of the required characteristics of the chosen problems was that they reflect real business use cases. Previous research has used datasets such as the NL4OPT dataset (Ramamonjison, Yu, Li, et al., 2023) to investigate whether or not LLMs are capable of formulating and solving mathematical optimization models (Ahmed & Choudhury, 2024; Xiao et al., 2024). But as mentioned by Ramamonjison, Yu, Li, et al. (2023) and other authors like Amarasinghe et al., 2023, a shortcoming of the dataset used in the NL4OPT competition is that it consists of only linear programming problems. Many real-world use cases require the introduction of integer variables or even include nonlinear terms. Additionally, problems included in the NL4OPT dataset are relatively simple, with only 2.08 variables and 2.83 constraints on average (Li et al., 2023). Both of these factors make problems included in the NL4OPT dataset unsuitable for the goal of this research as they do not accurately reflect the complexity of real business optimization problems. Since the release of the NL4OPT dataset, LLMs have also been tested on additional datasets. The dataset used by Li et al. (2023) builds upon the NL4OPT optimization problems. It introduces some logic constraints and integer variables to create a set of more complex, mixed integer programming problems. The NLP4LP dataset used in research by AhmadiTeshnizi et al. (2023) consists of struc-

tured natural language optimization problems in linear and mixed integer programming. Amarasinghe et al. (2023) investigate the abilities of a fine-tuned LLM at solving exclusively job shop scheduling problems. 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 *Claude 3 Opus* performs on different 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 classes. Given this thesis's time span and scope, 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 the four classes detailed in section 2.1 Mathematical Optimization: LP, MIP, IP, and 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 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. Li et al. (2023) also remark that the LLMs used in their study struggled with unit conversions. 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. Including both explicit and implicit natural language descriptions enhances the realism of the dataset. Business users may find it more intuitive to describe smaller decision-making problems using concrete values. However, for larger problems, 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, a bin packing problem, and an assignment problem. An example problem description for each optimization class can be found in section Example Problems in the Appendix, and an overview of each problem's characteristics can be found in section Overview of Dataset Characteristics in the Appendix. The problems were sourced from academic textbooks (Birge & Louveaux, 2011; Bracken & 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 & Tütüncü, 2006) and other online sources (MOSEK, 2024; Pedroso et al., 2019; StemEZ, n.d.). *Claude 3 Opus'* training data includes information up to August 2023 (Anthropic, 2024). As most data sources were readily available online, there was a

risk that some of the optimization problems were included in the LLM's training data. Consequently, the problems were altered by reformulating the natural language descriptions. Additionally, in some cases, *decision variables* were added, the *objective function* or *constraints* were modified, or some parameter values were changed. For most of the problems, no Python Pyomo code was available; hence, it was written by the project group members themselves. These factors help mitigate the risk of using optimization problems that the LLM has been trained over as part of this research. Overall, using a set of such diverse optimization problems aids in more realistically assessing whether *Claude 3 Opus* could be suitable for applications in real business settings, for example, as Decision Optimization CoPilot as envisioned by Wasserkrug et al. (2024).

3.2 Research Pipelines

Previous literature has experimented with different pipeline designs for the task of generating mathematical optimization problems with large language models (AhmadiTeshnizi et al., 2023; Fan et al., 2024; Amarasinghe et al., 2023; Li et al., 2023; Ramamonjison, Yu, Li, et al., 2023). Two pipelines of different complexities were used to conduct the experiments for this study to assess if a more advanced generation pipeline can improve the LLM's capabilities. Both approaches were designed to generate two main outputs: 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 in section 3.3 Evaluation Metrics. 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 and our familiarity with its syntax.

3.2.1 Pipeline 1

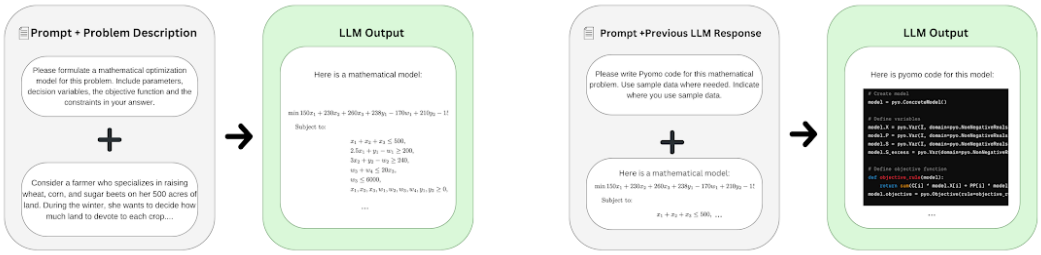
The idea for the first pipeline was to emulate a simple LLM query, in which the user asks to model their problem and attaches the problem description. Fan et al. (2024) took a similar approach in their experiment. This is also how a business user would likely interact with a contemporary LLM chat interface. The problem description was left unaltered and appended to the prompt, similar to the methodology outlined in Ramamonjison, Yu, Li, et al., 2023. 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. An example of running this step can be seen in Figure 3.1 (a).



(a) Model generation (b) Code generation

Figure 3.1: Model and code generation steps of pipeline 1

Next, with the first response of the LLM as input, *Claude 3 Opus* had to generate the Pyomo code corresponding to the mathematical formulation with the following prompt as seen in Figure 3.1 (b):

"Please write Pyomo code for this mathematical problem. Use sample data where needed. Indicate where you use sample data."

+

previous response

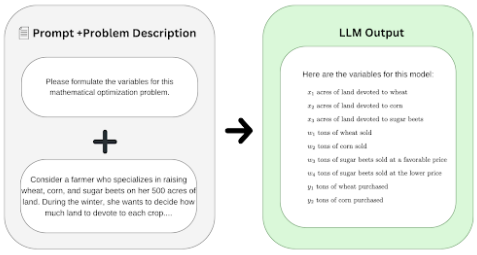
This was done regardless of the correctness of the mathematical model produced, as even an incorrect model can sometimes achieve the optimal solution, for example, due to the mistakes appearing in inactive constraints. The prompt also includes a statement allowing the model to use sample data where needed. 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. It was loaded manually for the problems requiring data, 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 to manually verify the viability of the generated mathematical formulation.

3.2.2 Pipeline 2

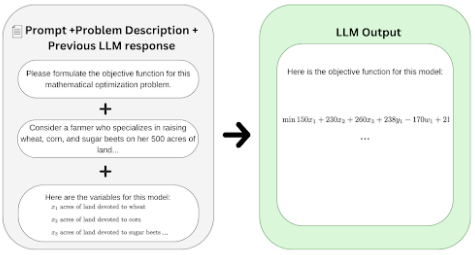
The second pipeline was designed to be a more advanced use of LLMs. The problem description input remained the same as in pipeline 1. However, the job of generating the model was split into three smaller tasks, following a chain-of-thought approach. This technique involves breaking down the task into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

Prompt 1	**Prompt 2**	**Prompt 3**
"Please formulate the variables for this mathematical optimization problem."	*"Please formulate the objective function for this mathematical optimization problem."*	*"Please formulate the constraints for this mathematical optimization problem."*
+	+	+
problem description	problem description	problem description
	+	+
	response to prompt 1	response to prompt 1
		+
		response to prompt 2

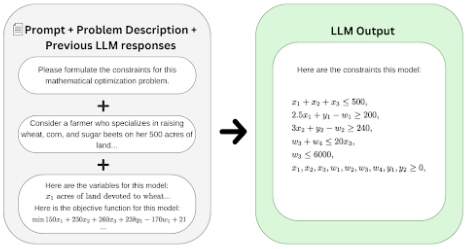
The three responses were then manually combined to form the full mathematical model. This approach can also be seen in Figure 3.2 (a) - (c). After this, the LLM was tasked to generate Pyomo code for the model in the same way as in pipeline 1, as seen in Figure 3.2 (d). A side-by-side comparison of both pipeline workflows is shown in Figure 3.3.



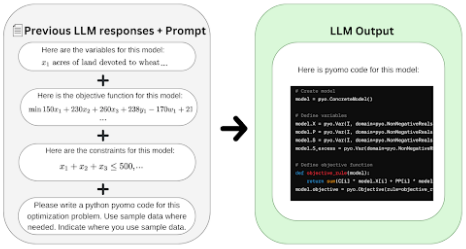
(a) Decision variable generation



(b) Objective function generation



(c) Constraints generation



(d) Code generation

Figure 3.2: Model and code generation steps of pipeline 2

Such division of work results in smaller sub-tasks, which might be easier for the LLM to solve, potentially leading to an improved outcome (Besta et al., 2024). On the other hand, formulating mathematical optimization problems step by step can take away some of the context. Some *decision variables*, for example, only become apparent when working on the *constraints* as some might require the introduction of helper variables such as logic constraints. The results of He et al. (2022) suggest that breaking down the task of synthesizing an optimization model from natural language input might adversely

affect results. Hence, this research aims to provide further insights into the effects of the two presented prompting techniques on the ability of *Claude 3 Opus* to formulate a correct mathematical model and code.

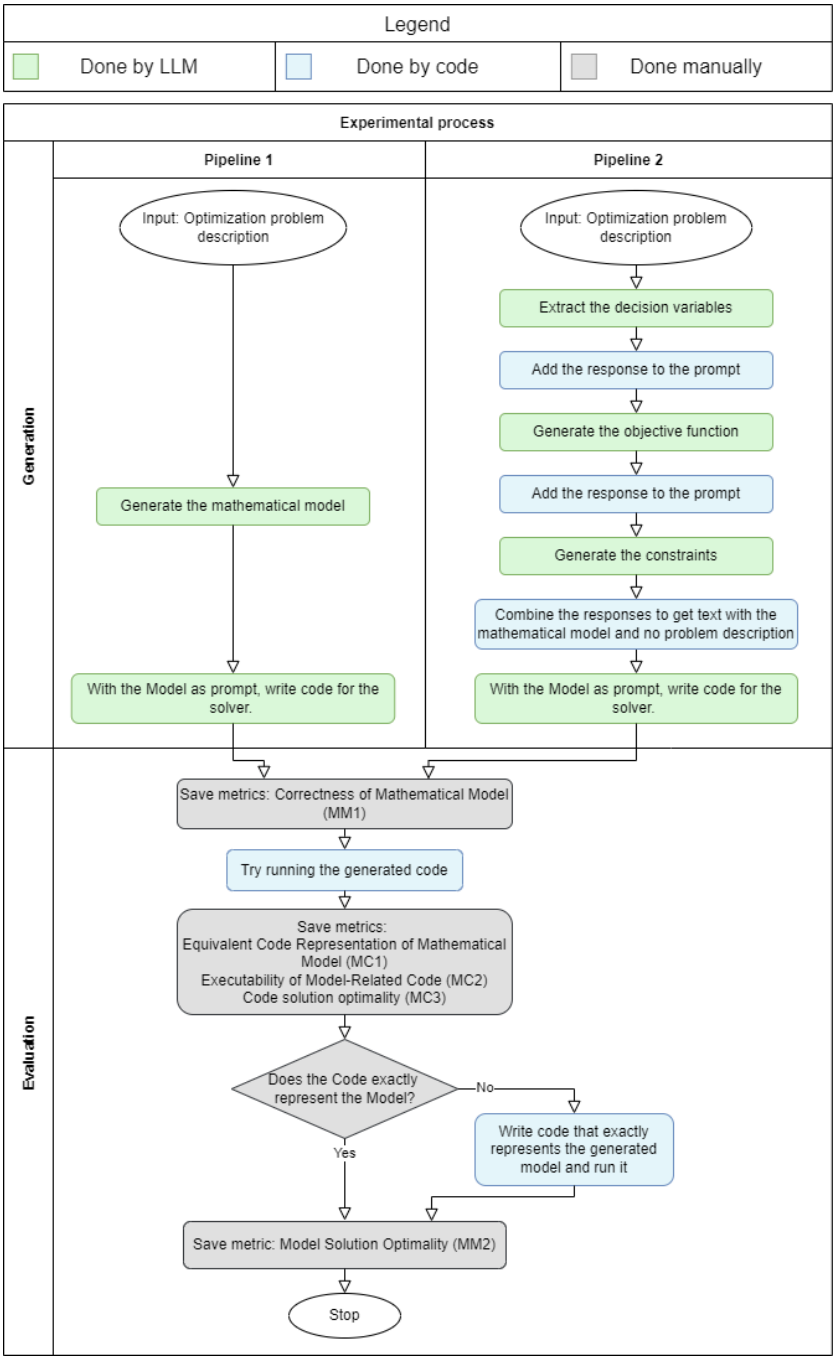


Figure 3.3: Modelling and evaluation workflow

3.3 Evaluation Metrics

Several metrics were defined to quantify the LLM's performance to correctly translate the problem description to a mathematical model, code, and solution. Generally, the metrics were divided into an evaluation of the generated mathematical model and an

evaluation of the code to solve the said model. All metrics were recorded manually at the stages presented in Figure 3.3.

Sarker et al. (2024) analyzed the ability of LLMs to produce code that solves mathematical equations. For this, they distinguish between the semantics and the syntax of the optimization problem. The semantics of two mathematical models represent their meaning and are considered 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 mathematical problem. Hence, two syntactically different models can still be semantically equivalent. Changing the syntax of a mathematical problem while preserving its semantics is called a *syntactic transformation*. Thus, the problem description, mathematical formulation, and code for a given problem are semantically equal but syntactically different. The model and component correctness metrics were measured by analyzing and comparing semantics between outputs and reference solutions. It is essential to make this distinction as any given optimization problem might be modeled in various ways, so there can be multiple semantically correct mathematical formulations per problem. As this research is not concerned with identifying whether or not LLMs can formulate the most efficient solution to a natural-language-based optimization problem, all semantically equivalent formulations are considered correct.

3.3.1 Correctness of Mathematical Model (MM_1)

This metric represents the LLM's ability to produce an entirely correct mathematical representation of the underlying problem (Li et al., 2023; Ramamonjison, Yu, Li, et al., 2023). To achieve this, the semantics between the mathematical model generated by the LLM and the reference solution were compared. To quantify this, the semantic equivalence of the *parameters* (P), *decision variables* (V), the *objective function* (O), and *constraints* (C) of the solution generated by LLM and the reference solution were measured. The binary metrics below were used for evaluation:

$$MM_1^{component}(M,R) = \begin{cases} 1, & \text{if } component \text{ of the mathematical model (M) semantically match } component \text{ of the reference solution (R)} \\ 0, & \text{otherwise} \end{cases}$$

$$\text{for } component \in \{\text{P, V, O, C}\}$$

$$MM_1(M,R) = \begin{cases} 1, & \text{if } MM_1^P, MM_1^V, MM_1^O, MM_1^C = 1 \\ 0, & \text{otherwise} \end{cases}$$

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*, *decision variables*, the *objective function*, or *constraints* at any of the steps, thus leading to ambiguous conclusions.

3.3.2 Model Objective Value Optimality (MM_2)

This metric measures if the objective value produced by the mathematical model of the LLM was optimal. Some mistakes could still lead to an optimal objective value depending on the problem instance, like additional inactive constraints or different variable domains. However, because the semantics of the formulations would be different, this optimal solution would not have been captured by MM_1. Thus, we defined the *Model Objective Value Optimality* as

$$MM_2(M, R) = \begin{cases} 1, & \text{if the objective value of the generated mathematical model} \\ & \text{(M) and reference solution (R) are equal} \\ 0, & \text{otherwise} \end{cases}$$

Assessing the objective value was chosen over assessing the optimality and feasibility of the optimal solution, as it will be the key number a decision-maker would use to guide their business strategies. Additionally, if the objective value is correct, so is the optimal solution in general, so using both metrics would be somewhat redundant.[2] There is a small possibility that the LLM generates a mathematical model with a mistake in the constraints that still results in the same objective value but with a solution that would be infeasible under the original constraints. However, this is not very likely, so the objective value optimality metric was deemed sufficient for exploring *Claude 3 Opus'* abilities and suitability as a potential Optimization CoPilot.

3.3.3 Model Consistency (MM_3)

In addition to the accuracy of the generated mathematical model and its solution, we were also interested in the consistency of the outputs. To be suitable for an automated optimization solution, *Claude 3 Opus* needs to generate correct solutions consistently, not simply by chance. To measure the consistency, the outputs were paired as a Cartesian product, and the number of agreeing output pairs[3] was divided by the total number of output pairs (Raj et al., 2022). Importantly, agreeing pairs do not mean semantically equivalent in this case but rather whether the LLM produced the same mathematical model. The reason for that lies in the model efficiency. While multiple semantically equivalent solutions can be correct, their efficiency might differ, leading to long computational times, especially in large models. For this reason, it is interesting to measure whether or not the LLM produces the same mathematical formulation when asked multiple times to generate the model. Thus, we achieve the model consistency metric through the following formula:

$$MM_3(M) = \frac{1}{n(n-1)} \sum_{i,j=1;i\neq j}^{n} \mathbb{1}(M_i = M_j)$$

[2]Feasability of the model will also be recorded but not used as a separate metric.

[3]The number of distinct permutations, not combinations, is used to count the amount of agreeing output pairs. Hence, every pair is counted twice.

where n = 3 is the number runs executed for a given problem, M_i is the ith generated mathematical model to a given problem, and $\mathbb{1}(M_i = M_j)$ is an indicator function for the mathematical equivalence.

3.3.4 Equivalence Code Representation of Mathematical Model (MC_1)

This metric represents the LLM's ability to apply a syntactic transformation to its generated mathematical formulation. Notably, it does not assess the correctness of the syntax of the code but simply the mathematical formulation within it. More formally, this metric is defined as

$$MC_1(M, C) = \begin{cases} 1, & \text{if the semantic meaning of the generated mathematical} \\ & \text{model (M) and the conversion into code (C) are equivalent} \\ 0, & \text{otherwise} \end{cases}$$

3.3.5 Executability of Model-Related Code (MC_2)

The code executability metric measures how well the LLM could use the chosen coding tools (Python and Pyomo) to arrive at a solution after providing a mathematical formulation. As was proposed by AhmadiTeshnizi et al. (2023) and Xiao et al. (2024), the generated model-related code[4] was tested for its executability and was formally defined as

$$MC_2(C) = \begin{cases} 1, & \text{if the model-related code executed (C) without errors} \\ 0, & \text{otherwise} \end{cases}$$

3.3.6 Code Objective Value Optimality (MC_3)

Finally, the *objective value* that resulted after a successful code execution measures the LLM's ability to autonomously arrive at the correct solution that was associated with the original problem description, as was seen in AhmadiTeshnizi et al. (2023). Notably, the sample data was manually replaced by the problem data related to the problem. The formula of this metric is:

$$MC_3(C, R) = \begin{cases} 1, & \text{if the } \textit{objective value} \text{ of the generated code (C) and reference} \\ & \text{solution (R) were equal} \\ 0, & \text{otherwise} \end{cases}$$

Thoroughly combining these metrics in the evaluation accurately describes the LLM's ability to solve optimization problems. It is important to note that all metrics except MM_3 were measured for each solving attempt and later aggregated. MM_3 was measured once for each problem.

[4]Model-related encompasses any Pyomo code used to define and solve the optimization model. Hence, any consecutive print statements the LLM uses to display the solution, for example, are excluded from this metric. Again, the reason for this lies in that accessing and displaying the optimal solution would be a software engineering task in an LLM-based optimization copilot.

4 Results

This section presents the results of the research conducted for this thesis. Generally, results for each individual problem are aggregated over the three trials. However, a detailed overview of the recorded results and each conversation with *Claude 3 Opus* is found in the shared GitHub[1] repository. Section 4.1 analyzes the LLM's general abilities to translate business problems from natural-language descriptions into mathematical models and executable code. Section 4.2 analyzes whether there are significant differences in the LLM's capabilities in formulating and solving optimization problems of different optimization classes as well as complexity levels. Throughout both sections, the two approaches used for the experiments, *pipeline 1* and *pipeline 2*, will be compared. Unless stated otherwise, the differences presented are not statistically significant at $\alpha = 0.05$.[5]

4.1 General Results Comparison

4.1.1 Mathematical Model Evaluation

Table 4.1 displays the overall success rates of *Claude 3 Opus* regarding the task of synthesizing a correct mathematical model from a natural language problem description and reaching the optimal objective value. All values displayed in the tables are aggregated over the whole dataset and shown as percentages and raw counts of successes.

Table 4.1: Correctness of Mathematical Model (MM_1) & Model Obj. Value Optimality (MM_2)

	Parameters	Variables	Objective	Constraints	Model	Solution
P1	89.58% (43)	83.33% (40)	85.42% (41)	43.75% (21)	41.67% (20)	47.92% (23)
P2	81.25% (39)	89.58% (43)	75.00% (36)	50.00% (24)	45.83% (22)	54.17% (26)
Avg	85.42%	86.46%	80.21%	46.88%	43.75%	51.05%

In *pipeline 1*, *Claude 3 Opus* more often correctly defined parameters and the objective function. In *pipeline 2*, *Claude 3 Opus* more often correctly defined variables, constraints, and the mathematical model. *Pipeline 2* resulted in the correct objective value more frequently.

[5]Section Statistical Significance of Results in the Appendix contains a detailed overview of the p-values for the differences in the metrics between *pipeline 1* and *pipeline 2*, optimization classes and constraint complexity levels. A paired t-test was used to determine the significance of differences between the two pipelines. An independent t-test was used to determine the significance of differences within a pipeline between optimization classes and constraint complexities.

Overall, *pipeline 2* achieved slightly better total model correctness of 45.83% compared to 41.67% in *pipeline 1*. Only 4.17% of the optimization models generated by *Claude 3 Opus* in *pipeline 1* were infeasible and 6.25% in *pipeline 2*. In terms of modeling each component of the mathematical model, the results in Table 4.1 show that, on average, *Claude 3 Opus'* correctly models parameters, decision variables, and objectives in more than 80% of cases. The average constraint correctness, on the other hand, is only 46.88%. The difference between parameter and constraint correctness as well as variable and constraint correctness is statistically significant in both pipelines. In *pipeline 1*, the objective and constraint correctness is also statistically different. Regardless of the pipeline, *Claude 3 Opus* commonly failed to correctly model complex constraints such as logic, robust, or time-dynamic constraints. A more detailed discussion about *Claude 3 Opus'* ability to model problems containing such complex constraints can be found in section 4.2.2. The LLM also had issues correctly defining constraints and objective functions consistently when the problem description was somewhat ambiguous and did not mention the mathematical logic explicitly. For example, in LP2 the storage constraint was described as:

"Your company can also only store a limited amount of raw materials."

Reference Solution	**Claude 3 Opus' Solution**
$r_1 + r_2 \leq \mathrm{S}$	$r_1 \leq \mathrm{S}_1$
	$r_2 \leq \mathrm{S}_2$

A human expert could clarify with the decision-maker what is meant in such cases, but the LLM simply chooses the option with the highest probability given the context and the data on which it was trained. Finally, in some cases (IP3, NLP1), *Claude 3 Opus* modeled a capacity constraint that was never mentioned in the problem description. A possible explanation for this is that in many production planning and assignment optimization problems, one of the key constraints is the limited availability of one of the resources. Due to being trained on such problems, *Claude 3 Opus* might have been biased to include this constraint even if the problem at hand does not mention any limitation on resource availability. The lack in the LLM's ability to correctly model the constraints seems to be the main driver for the overall limited success in correctly translating the problem description into a mathematical model, as indicated by a correlation of 0.98 and 0.94 between constraint correctness and model correctness in *pipeline 1* and *pipeline 2*, respectively. Altogether, the mathematical models formulated by *Claude 3 Opus* yielded an optimal solution in 47.92% of cases in *pipeline 1* and 54.17% in *pipeline 2* (see Table 4.1). These values are slightly higher than the model correctness. This is because, in some cases, the mistakes made by *Claude 3 Opus* did not affect the objective value of the solution. For example, in one of the solutions to LP4, the LLM used an equality constraint instead of the correct inequality constraint.

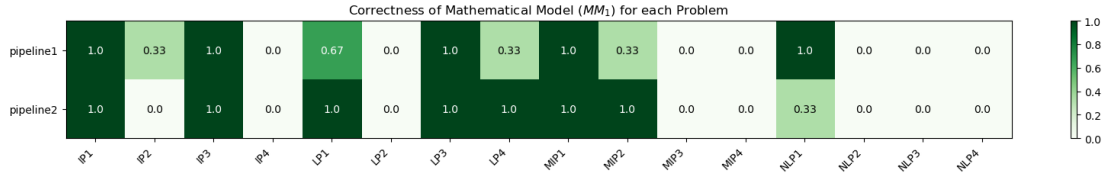


Figure 4.1: Breakdown of Correctness of Mathematical Model (MM_1) across each problem in the dataset for all three trials in both pipelines: correct 0 times (0.00), correct 1 time (0.33), correct 2 times (0.67), or correct all 3 times (1.00).

Another interesting observation can be seen in Figure 4.1, which displays a breakdown of the model correctness metric, MM_1, across all problems included in the dataset. With only one exception, IP2, each problem was either modeled wrong across all three trials, the white squares, or at least once successfully, the green squares, in both pipelines. There might be an underlying characteristic to the problems the LLM consistently failed to model, which increases *Claude 3 Opus'* likelihood of making a mistake. Indeed, except for problem NLP2, all problems the LLM consistently modeled incorrectly contained complex constraints. Conversely, only one of the problems for which the LLM succeeded at least once, IP3, contained complex constraints. Figure 4.1 additionally suggests that using *pipeline 2*, *Claude 3 Opus* might formulate a correct mathematical model more reliably than using *pipeline 1*. In *pipeline 2*, only in one case, NLP1, the model correctness was not equal to either 0 or 1, meaning that either in none of the three trials or in all of the three trials was the problem formulated correctly. In *pipeline 1*, this was the case for four problems, IP2, LP1, LP4, and MIP2.

4.1.2 Code Evaluation

Table 4.2 displays the three code evaluation metrics aggregated over the whole dataset: Equivalence Code Representation of Mathematical Model (MC_1), Executability of Model-Related Code (MC_2) and Code Objective Value Optimality (MC_3).

Table 4.2: Code Evaluation Metrics

	Translation	Executability	Solution
P1	93.75% (45)	72.92% (35)	41.67% (20)
P2	85.42% (41)	62.50% (30)	45.83% (22)
Avg	89.59%	67.71%	43.75%

In *pipeline 1*, *Claude 3 Opus* more often correctly translated the mathematical model into Pyomo code and wrote executable code. *Pipeline 2* resulted in the correct objective value more frequently.

Using the one-step prompting technique of *pipeline 1*, *Claude 3 Opus* could better translate the mathematical model into Pyomo code indicated by the success rate of 93.75% compared to 85.42% in *pipeline 2*. A possible reason for this could be differences in the LLM's outputs in the modeling part of the pipeline. The average number of output tokens per API call in *pipeline 1* was 964, while in *pipeline 2*, it was 540. However, three API calls were necessary to generate the mathematical formulation of the problem

in *pipeline 2*, so the average number of output tokens for the complete mathematical formulation is 1620. As the underlying mathematical model would require a similar amount of tokens to model, this suggests that *Claude 3 Opus* added additional information in the second pipeline, such as verbal explanations of the model's components, which might hinder the translation from a mathematical model into Python Pyomo code.

In *pipeline 1*, the code executability rate was higher than in *pipeline 2* as well. Figure 4.2, a breakdown of the code executability rate across all problems in the dataset, also suggests that whether or not *Claude 3 Opus* was able to formulate an executable Pyomo code was unrelated to the underlying optimization problem. Only in half of the studied problems do the executability rates match across the two pipelines.

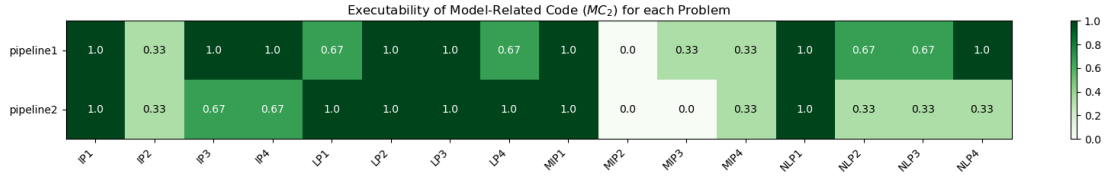


Figure 4.2: Breakdown of Executability of Model-Related Code (MC_2) across each problem in the dataset for all three trials in both pipelines: executable 0 times (0.00), executable 1 time (0.33), executable 2 times (0.67), or executable all 3 times (1.00).

In both pipelines, the executability problems can be generally grouped into three categories: incorrect use of solver, incorrect use of Pyomo syntax, and incorrect variable indexing. Interestingly, solver issues mainly occurred when the model incorrectly used non-linear terms in otherwise linear problems (LP1, IP2), as the LLM used a solver suited for solving linear optimization problems. This happened even though *Claude 3 Opus* was not provided with any information about the class of optimization problem at hand. Common issues regarding incorrect use of Pyomo syntax were the use of logic operators in objective functions and constraints (LP4, MIP2, NL3) and using ranged bounds in constraints instead of separate expressions for upper and lower bounds (MIP3, NL3, NL4). *Claude 3 Opus* also made a mistake applying the Pyomo *summation* function (MIP2) as well as incorrectly using a built-in math function as opposed to using Pyomo math functions (NL2). Finally, on two occasions, the LLM incorrectly accessed indexed variables (MIP2, MIP4).

For *pipeline 1* and *pipeline 2*, the code objective value optimality is slightly lower than the model objective value optimality at 41.67% and 45.83%, respectively. This is due to the non-executability of some problems' code and wrong model translations. In neither of the pipelines did the LLM produce a correct code solution if the mathematical model solution was incorrect, which was expected due to the set-up of the code-generation prompt.

4.1.3 Consistency Evaluation

Lastly, the consistency of the mathematical models generated by *Claude 3 Opus* was assessed and is depicted in Table 4.3. The consistency in the LLM's answers across the whole dataset was 35.31% in *pipeline 1* and 33.25% in *pipeline 2*. So, on average, the LLM produced two mathematically matching models and one differing model over the three runs per problem.

Table 4.3: Model Consistency (MM_3)

	Consistency
Pipeline1	35.31%
Pipeline2	33.25%
Avg	34.28%

In *pipeline 1*, *Claude 3 Opus* generated consistent mathematical models slightly more often than in *pipeline 2*.

As with the code executability, the consistency in *Claude 3 Opus'* answers does not seem to relate strongly to the underlying problem, as only for seven of the 16 problems does MM_3 match across both pipelines as seen in Figure 4.3. Additionally, the LLM generally generates multiple mathematically different models independently of whether or not it can correctly model the optimization problem. In fact, there is only a low positive correlation between consistency and the model correctness (MM_1) of 0.43 in *pipeline 1* and 0.35 in *pipeline 2*.

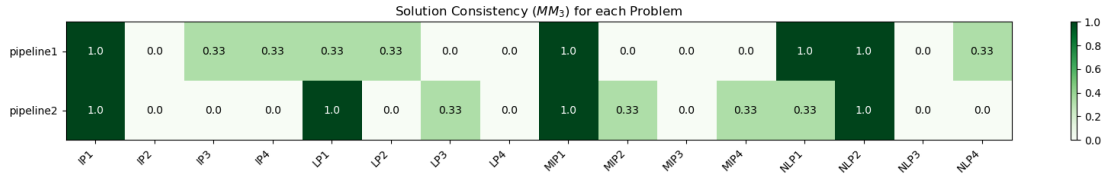


Figure 4.3: Breakdown of Model Consistency (MM_3) across each problem in the dataset in both pipelines: 3 distinct mathematical models (0.00), 2 distinct mathematical models (0.33), or 1 distinct mathematical model (1.00).

4.2 Section 2

4.2.1 Impact of Problem Class on Performance

This subsection will investigate whether there are significant differences in *Claude 3 Opus'* abilities at modeling and solving optimization problems of the four different classes studied in this research: IP, LP, MIP, and NLP. Figure 4.4 shows the average model correctness per optimization class across the two different pipelines. Using the one-step prompting technique of *pipeline 1*, the LLM achieved the highest model correctness of 0.58 for IPs, followed by 0.50 for LPs, 0.33 for MIPs, and 0.25 for NLPs.

However, the differences between the results for each optimization class are not statistically significant. With the multi-step prompting approach of *pipeline 2*, *Claude 3 Opus* correctly modeled LPs in 75% of cases. LPs are followed by IPs and MIPs, both with a model correctness of 0.50, and NLPs with 0.08. In contrast to *pipeline 1*, there is a statistically significant difference in the LLM's ability to model problems of classes LP and NLP in *pipeline 2*, as indicated by a p-value of 0.0447. In both pipelines, problems of class NLP were modeled correctly least often.

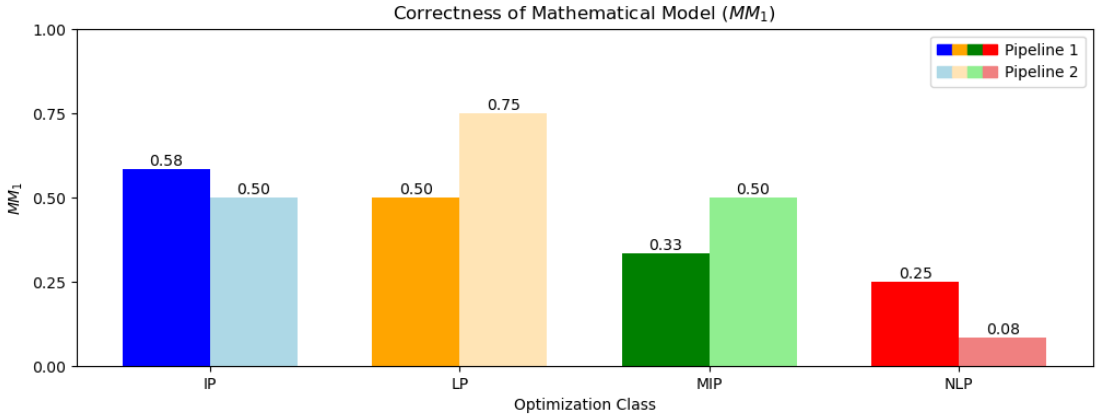


Figure 4.4: Correctness of Mathematical Model (MM_1) per optimization class. In *pipeline 1*, *Claude 3 Opus* was best able to model IPs, then LPs, then MIPs, and finally NLPs. In *pipeline 2*, *Claude 3 Opus* was best able to model LPs, then IPs and MIPs, and finally NLPs.

Each of the pipelines outperformed the other in two of the optimization categories. In *pipeline 1*, *Claude 3 Opus* achieved better model correctness for IPs and NLPs, while in *pipeline 2*, it performed better for LPs and MIPs. However, there is no statistically significant difference between the two pipelines in modeling different optimization classes.

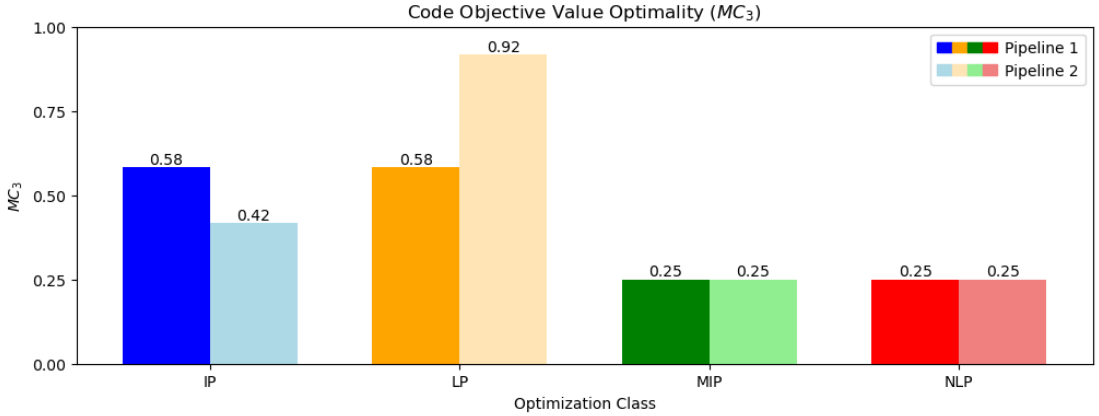


Figure 4.5: Code Objective Value Optimality (MC_3) per optimization class. In *pipeline 1*, *Claude 3 Opus* was best able to solve IPs and LPs, and then MIPs and NLPs using Pyomo in Python. In *pipeline 2*, *Claude 3 Opus* was best able to solve LPs, then IPs, and then MIPs and NLPs using Pyomo in Python.

Figure 4.5 displays the code objective value optimality, which reflects *Claude 3 Opus'* ability to solve optimization problems by correctly translating them into executable Python Pyomo code. For both pipelines, the trends are similar to the model correctness. This result is logical as the code objective value optimality largely depends on the correctness of the underlying mathematical model. The LLM performed best for IPs and LPs with a code objective value optimality of 0.58, followed by MIPs and NLPs with a code objective value optimality of 0.25 in *pipeline 1*. In *pipeline 2*, LPs achieved the highest code objective value optimality of 0.92, followed by IPs with 0.42 and MIPs and NLPs with 0.25. Only the difference in performance between LPs and MIPs as well as LPs and NLPs in *pipeline 2* is statistically significant. As explained in section 4.1, differences between model correctness and code objective value optimality mainly stem from two kinds of mistakes: generating inexecutable code, leading to a lower MC_3 than MM_1, and model component mistakes that do not negatively impact the solution, leading to a higher MC_3 than MM_1. These mistakes occur rarely, and there is no statistically significant difference in the frequency of these mistakes between the optimization classes.

4.2.2 Impact of Constraint Complexity on Performance

Finally, this subsection looks into potential differences in *Claude 3 Opus* ability to model problems with constraints of two different complexity levels, regular and difficult. Problems with difficult constraints contained either logic, robust, or time-dynamic constraints. Figure 4.6 shows that in both pipelines, the LLM can correctly model problems with regular constraints around twice as often as problems with difficult constraints.

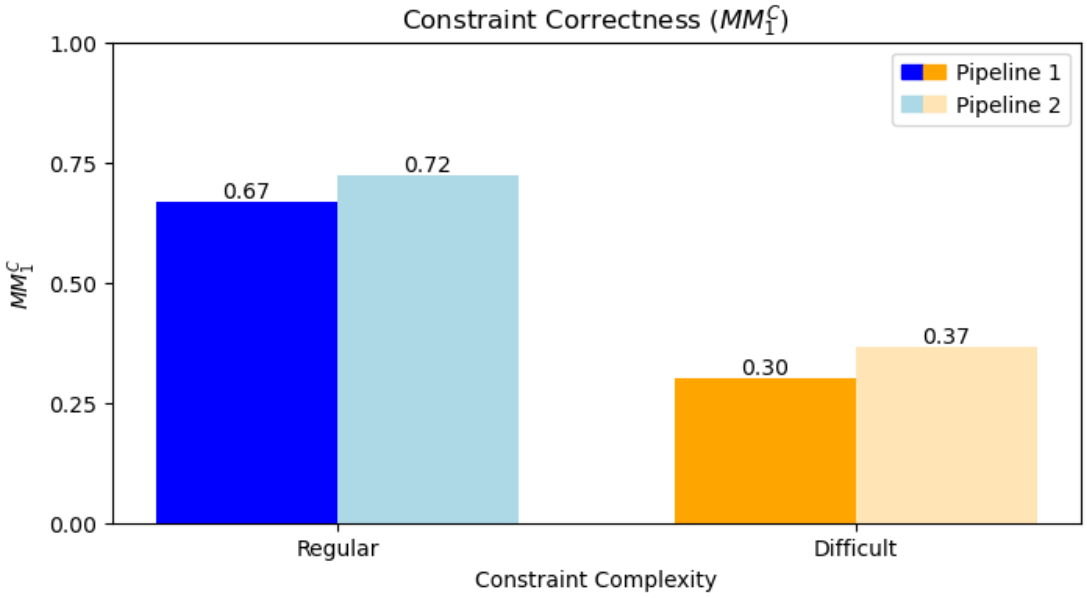


Figure 4.6: Constraint Correctness (MM_1^C) per complexity level. In both pipelines, *Claude 3 Opus* correctly models regular constraints more frequently than complex constraints.

Under the one-step approach of *pipeline 1*, this is indicated by the constraint correctness of 0.67 for problems with regular constraints and 0.30 for problems with difficult constraints. Using the multi-step technique of *pipeline 2*, *Claude 3 Opus* achieved constraint correctness of 0.72 for problems with regular constraints and 0.37 for problems with difficult constraints. For logic and robust constraints, the LLM most commonly simply ignored the logic or robustness. One example of ignored constraints can be seen in problem IP4, which includes the following logic constraint:

"You might sell fewer (negative deviation) or more (positive deviation) than this target."

Reference Solution	**Claude 3 Opus' Solution**
$5X_1 + 2X_2 + d_1^- - d_1^+ = 5500$	$5X_1 + 2X_2 + d_1^- - d_1^+ = 5500$
$d_1^- \leq MY_{11}$	
$d_1^+ \leq MY_{12}$	
$Y_{11} + Y_{12} = 1$	

With time-dynamic constraints, *Claude 3 Opus* was often able to correctly model the constraint that defines the relationships between two consecutive states. However, it ignored the need to define an initial zero-state, making the mathematical model incomplete.

5 Conclusion and Discussion

Overall, *Claude 3 Opus* has shown a moderate ability to translate business optimization problems into correct mathematical models. No statistically significant difference existed between performance on different optimization classes or complexity levels. It only defined a correct mathematical formulation in slightly less than half of the cases for both pipelines studied. However, the LLM often only made small mistakes in wrong models, such as using equality instead of inequality in the constraints. The metrics used throughout this study are binary due to their implications for using LLMs as part of potential Optimization CoPilots - to formulate business strategies, a decision maker needs to be able to rely on the results provided by the LLM to be fully accurate. However, this also means they might not appropriately capture the degree to which *Claude 3 Opus* is actually able to formulate mathematical optimization problems. Future research could address this by using more nuanced metrics, such as an accuracy metric for the model correctness as used by Ramamonjison, Yu, Li, et al. (2023) instead of the binary correctness metric. Another option would be to measure the distance of the solution of the model provided by the LLM to the reference solution.

While the LLM has shown a good ability to extract *parameters*, *decision variables*, and *objective functions* from natural-language-based problem descriptions, it commonly failed to model constraints, especially complex constraints, correctly. This is a key insight of this research and can be used to improve the performance of *Claude 3 Opus* and other LLMs concerning optimization modeling. Li et al. (2023) showed first successes in using a constraint classifier to improve *GPT-3.5's* and *PaLM 2's* optimization modelling performance. A similar classifier could be used as part of an Optimization CoPilot. Another suggestion would be to combine such a classifier with in-prompt examples of correctly modeled constraints for each constraint class, as a lot of research has shown initial success using in-context learning to improve LLMs' performance on a wide range of tasks (Brown et al., 2020; Wei et al., 2022; Zhao et al., 2021). Additionally, *Claude 3 Opus* had problems correctly modeling constraints when the problem description was not very explicit. In a real Optimization CoPilot, the LLM would likely encounter similarly ambiguous input, even if the user is asked to be clear in their descriptions. Future research could experiment with pipeline designs that allow the LLM to clarify ambiguous problem descriptions, emulating the interaction between OR experts and decision-makers in such cases. Finally, LLMs' tendency for hallucinations extends to optimization modeling: for two of the sixteen problems, *Claude 3 Opus* included

constraints in the mathematical models that are common for the type of optimization problem at hand but were not actually part of the business problem.

Generally, *Claude 3 Opus* was relatively consistent in its answers, generating, on average, two mathematically identical models per problem tested. To be used reliably as part of an Optimization CoPilot, it would be interesting to investigate ways to make the LLM's answers even more consistent in future research. This could include modifications to the prompts as well as modifications of LLM-specific parameters, such as temperature, which is an indicator of the LLM's creativity setting.[6]

Regarding translating mathematical models into executable code, *Claude 3 Opus* generally has shown a good ability to syntactically transform models into Python Pyomo code. The LLM was also moderately skilled at writing executable Python Pyomo code, mainly failing with variable ranged bounds, choosing the appropriate solver, and some Pyomo-specific syntax. Further improvements for using *CLaude 3 Opus* as part of an Optimization CoPilot could be made by defining specific rules as part of the prompt that the LLM should follow when writing Pyomo code. The prompt could, for example, include the rule that ranged bounds need to be modeled as two separate constraints, one for the lower bound and one for the upper bound. Another option would be to let the LLM define the mathematical model in a standardized output format, such as LaTeX, and then use a parser as suggested by Ramamonjison, Yu, Xing, et al. (2023) to translate this into Pyomo code instead of relying on the LLM for this task.

Besides *Claude 3 Opus* ability to formulate and solve mathematical optimization models of varying problem classes and complexity levels from natural-language descriptions, there are two other concerns for designing an Optimization CoPilot. Firstly, costs play an important role in designing an Optimization CoPilot. This research has demonstrated that *pipeline 2* achieved better results in correctly modeling optimization problems, but they were not statistically significant. *Pipeline 2* also required significantly more input and output tokens. At $15 per one million input tokens and $75 per one million output tokens, *Claude 3 Opus* is one of the most expensive LLMs available. When designing an Optimization CoPilot, it is important to consider whether the potential improvement in model correctness in *pipeline 2* justifies the increased costs. Secondly, this research design does not provide any way of handling incorrect solutions. *Claude 3 Opus* reached a solution in around 68% of cases, but only around 65% of those generated answers were correct. So, with the current design, the business user would have received an incorrect solution in around one-third of cases without being aware of that. As these solutions might be used to guide vital business decisions, an actual Optimization CoPilot must include some tests and verifications to be able to guarantee correct solutions. One potential approach would be to include self-correction of the LLM via automated feedback in the pipeline, as previous research has shown that it can reduce

[6]For this research, the *Claude 3 Opus'* has been left at its default of 1.0, meaning most creative freedom.

errors on various tasks (Pan et al., 2023). Future research could look into the effects of the ability to self-correct on the results presented in this thesis, especially focusing on the frequency of incorrect solutions.

In conclusion, the answer to the question **"How well are cutting-edge LLMs suited for the task of formulating and solving mathematical optimization models of varying problem classes and complexity levels form natural-language descriptions?"** posed in chapter 1 is that the state-of-the-art LLM *Claude 3 Opus* shows potential at formulating and solving various kinds of optimization models. However, it cannot yet serve as an autonomous Optimization CoPilot due to problems with accurately modeling constraints and writing executable Python Pyomo code. So, further research and pipeline improvements are needed to design a set-up that will work in practice.

References

Abdin, A. F., Fang, Y.-P., Caunhye, A., Alem, D., Barros, A., & 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. https://doi.org/10.1016/j.ejor.2021.10.062

Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., et al. (2023). Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*. https://doi.org/10.48550/arXiv.2303.08774

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

Ahmed, T., & Choudhury, S. (2024). Lm4opt: Unveiling the potential of large language models in formulating mathematical optimization problems. *arXiv preprint arXiv: 2403.01342*. https://doi.org/10.48550/arxiv.2403.01342

Amarasinghe, P. T., Nguyen, S., Sun, Y., & 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. (2023). *Model card and evaluations for claude models* (tech. rep.). Anthropic. Retrieved May 30, 2024, from https://www-cdn.anthropic.com/bd2a28d2535bfb0494cc8e2a3bf135d2e7523226/Model-Card-Claude-2.pdf

Anthropic. (2024). *The claude 3 model family: Opus, sonnet, haiku* (tech. rep.). Anthropic. Retrieved May 30, 2024, from https://www-cdn.anthropic.com/de8ba9b01c9ab7cbabf5c33b80b7bbc618857627/Model_Card_Claude_3.pdf

Besta, M., Memedi, F., Zhang, Z., Gerstenberger, R., Piao, G., Blach, N., Nyczyk, P., Copik, M., Kwaśniewski, G., Müller, J., Gianinazzi, L., Kubicek, A., Niewiadomski, H., O'Mahony, A., Mutlu, O., & Hoefler, T. (2024). Demystifying chains, trees, and graphs of thoughts. *arXiv preprint arXiv:2401.14295*. https://doi.org/10.48550/arXiv.2401.14295

Birge, J. R., & Louveaux, F. (2011). *Introduction to stochastic programming*. Springer Science & Business Media. https://doi.org/10.1007/978-1-4614-0237-4

Bracken, J., & McCormick, G. P. (1968). *Selected applications of nonlinear programming*. Wiley New York.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. (2020). Language models are few-

shot learners. *Advances in neural information processing systems*, *33*, 1877–1901. https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf

Castillo, E., Conejo, A. J., Pedregal, P., Garcia, R., & Alguacil, N. (2011). *Building and solving mathematical programming models in engineering and science*. John Wiley & Sons. https://doi.org/10.1007/978-3-030-97626-2

Chen, M., Tworek, J., Jun, H., Yuan, Q., de Oliveira Pinto, H. P., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., Ray, A., Puri, R., Krueger, G., Petrov, M., Khlaaf, H., Sastry, G., Mishkin, P., Chan, B., Gray, S., … Zaremba, W. (2021). Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*. https://doi.org/10.48550/arXiv.2107.03374

Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., et al. (2021). Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*. https://doi.org/10.48550/arXiv.2110.14168

Cornuejols, G., & Tütüncü, R. (2006). *Optimization methods in finance* (Vol. 5). Cambridge University Press. https://doi.org/10.1017/9781107297340

Cornuéjols, G., Peña, J., & Tütüncü, R. (2018a). Linear programming: Theory and algorithms. In *Optimization methods in finance* (pp. 11–34). Cambridge University Press.

Cornuéjols, G., Peña, J., & Tütüncü, R. (2018b). Mixed integer programming: Theory and algorithms. In *Optimization methods in finance* (pp. 140–160). Cambridge University Press.

Cornuéjols, G., Peña, J., & Tütüncü, R. (2018c). Nonlinear programming: Theory and algorithms. In *Optimization methods in finance* (pp. 305–320). Cambridge University Press.

Dantzig, G. B. (1990). Origins of the simplex method. In *A history of scientific computing* (pp. 141–151).

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*. https://doi.org/10.48550/arXiv.1810.04805

Fan, Z., Ghaddar, B., Wang, X., Xing, L., Zhang, Y., & Zhou, Z. (2024). Artificial intelligence for operations research: Revolutionizing the operations research process. *arXiv preprint arXiv:2401.03244*. https://doi.org/10.48550/arxiv.2401.03244

Gangwar, N., & Kani, N. (2023). Highlighting named entities in input for auto-formulation of optimization problems. *arXiv preprint arXiv:2212.13201*. https://doi.org/10.48550/arXiv.2212.13201

Gurobi Optimization, LLC. (n.d.). *Gurobi Optimizer*. Retrieved May 26, 2024, from https://www.gurobi.com/solutions/gurobi-optimizer/

Hart, W. E., Watson, J.-P., & Woodruff, D. L. (2011). Pyomo: Modeling and solving mathematical programs in python. *Mathematical Programming Computation*, *3*(3), 219–260. https://doi.org/10.1007/s12532-011-0026-8

He, J., Vignesh, S., Kumar, D., Uppal, A., et al. (2022). Linear programming word problems formulation using ensemblecrf ner labeler and t5 text generator with data augmentations. *arXiv preprint arXiv:2212.14657*. https://doi.org/10.48550/arXiv.2212.14657

Hendrycks, D., Burns, C., Kadavath, S., Arora, A., Basart, S., Tang, E., Song, D., & Steinhardt, J. (2021). Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*. https://doi.org/10.48550/arXiv.2103.03874

Huang, Y., Zhang, W., Feng, L., Wu, X., & Tan, K. C. (2024). How multimodal integration boost the performance of llm for optimization: Case study on capacitated vehicle routing problems. *arXiv preprint arXiv:2403.01757*. https://doi.org/10.48550/arXiv.2403.01757

Jang, S. (2022). Tag embedding and well-defined intermediate representation improve auto-formulation of problem description. *arXiv preprint arXiv:2212.03575*. https://doi.org/10.48550/arXiv.2212.03575

Land, A. H., & Doig, A. G. (1960). An automatic method of solving discrete programming problems. *Econometrica*, *28*(3), 497–520. https://doi.org/10.2307/1910129

Lasdon, L. S., Waren, A. D., Jain, A., & Ratner, M. (1978). Design and testing of a generalized reduced gradient code for nonlinear programming. *ACM Transactions on Mathematical Software (TOMS)*, *4*(1), 34–50. https://doi.org/10.1145/355769.355773

Le, H., Wang, Y., Gotmare, A. D., Savarese, S., & Hoi, S. C. H. (2022). CodeRL: Mastering code generation through pretrained models and deep reinforcement learning. *Advances in Neural Information Processing Systems*, *35*, 21314–21328.

Lewis, M., Liu, Y., Goyal, N., Ghazvininejad, M., Mohamed, A., Levy, O., Stoyanov, V., & Zettlemoyer, L. (2019). Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*. https://doi.org/10.48550/arXiv.1910.13461

Li, Q., Zhang, L., & Mak-Hau, V. (2023). Synthesizing mixed-integer linear programming models from natural language descriptions. *arXiv preprint arXiv:2311.15271*. https://doi.org/10.48550/arXiv.2311.15271

Mistral AI. (2024, April). *Cheaper, better, faster, stronger: Continuing to push the frontier of ai and making it accessible to all*. Retrieved May 29, 2024, from https://mistral.ai/news/mixtral-8x22b/

MOSEK. (2024). *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., Barnes, N., & Mian, A. (2023). A comprehensive overview of large language models. *arXiv preprint arXiv:2307.06435*. https://doi.org/10.48550/arXiv.2307.06435

Ning, Y., Liu, J., Qin, L., Xiao, T., Xue, S., Huang, Z., Liu, Q., Chen, E., & Wu, J. (2023). A novel approach for auto-formulation of optimization problems. *arXiv preprint arXiv:2302.04643*. https://doi.org/10.48550/arXiv.2302.04643

Omran, S. M., El-Behaidy, W. H., & Youssif, A. A. (2023). Optimization of cryptocurrency algorithmic trading strategies using the decomposition approach. *Big Data and Cognitive Computing*, *7*(4), 174. https://doi.org/10.3390/bdcc7040174

OpenAI. (2023). *Data Analyst*. Retrieved May 26, 2024, from https://chatgpt.com/g/g-HMNcP6w7d-data-analyst

Pan, L., Saxon, M., Xu, W., Nathani, D., Wang, X., & Wang, W. Y. (2023). Automatically correcting large language models: Surveying the landscape of diverse self-correction strategies. *arXiv preprint arXiv:2308.03188*. https://doi.org/10.48550/arXiv.2308.03188

Pedroso, J. P., Rais, A., Kubo, M., & Muramatsu, M. (2019). *Facility location problems – mathematical optimization: Solving problems using gurobi and python*. Retrieved May 23, 2024, from https://scipbook.readthedocs.io/en/latest/flp.html

Poler, R., Mula, J., & Díaz-Madroñero, M. (2014). *Operations research problems*. Springer eBooks. https://doi.org/10.1007/978-1-4471-5577-5

Prompt Shell Smith. (2023). *Code Copilot*. Retrieved May 26, 2024, from https://chatgpt.com/g/g-2DQzU5UZl-code-copilot

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. (2019). Language models are unsupervised multitask learners. *OpenAI blog*, *1*(8), 9.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., & Liu, P. J. (2020). Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, *21*(140), 1–67.

Raj, H., Rosati, D., & Majumdar, S. (2022). Measuring reliability of large language models through semantic consistency. *arXiv preprint arXiv:2211.05853*. https://doi.org/10.48550/arXiv.2211.05853

Ramamonjison, R., Yu, T. T., Li, R., Li, H., Carenini, G., Ghaddar, B., He, S., Mostajabdaveh, M., Banitalebi-Dehkordi, A., Zhou, Z., & Zhang, Y. (2023). Nl4opt competition: Formulating optimization problems based on their natural language descriptions. *arXiv preprint arXiv:2303.08233*. https://doi.org/10.48550/arXiv.2303.08233

Ramamonjison, R., Yu, T. 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 D. Bollegala, R. Huang, & A. Ritter (Eds.), *Proceedings of the 61st annual meeting of the association for computational linguistics (volume 3: Sys-*

tem demonstrations) (pp. 471–478). Association for Computational Linguistics. https://doi.org/10.18653/v1/2023.acl-demo.45

Reid, M., Savinov, N., Teplyashin, D., Lepikhin, D., Lillicrap, T., Alayrac, J.-b., Soricut, R., Lazaridou, A., Firat, O., Schrittwieser, J., et al. (2024). Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530*. https://doi.org/10.48550/arXiv.2403.05530

Sakaguchi, K., Bras, R. L., Bhagavatula, C., & Choi, Y. (2021). Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, *64*(9), 99–106. https://doi.org/10.1145/3474381

Sarker, L., Downing, M., Desai, A., & Bultan, T. (2024). Syntactic robustness for llm-based code generation. *arXiv preprint arXiv:2404.01535*. https://doi.org/10.48550/arXiv.2404.01535

Sider. (2023). *Scholar GPT*. Retrieved May 26, 2024, from https://chatgpt.com/g/g-kZ0eYXlJe-scholar-gpt

Singh, A. (2012). An overview of the optimization modelling applications. *Journal of Hydrology*, *466*, 167–182. https://doi.org/10.1016/j.jhydrol.2012.08.004

StemEZ. (n.d.). *Problem 04 – 0177 — stem ez*. Retrieved May 23, 2024, from https://stemez.com/subjects/science/1HOperationsReseach/1HOperationsReseach/1HOperationsResearch/1H04-0177.htm

Thompson, A. D. (2024, March). *The memo - special edition: Claude 3 opus*. Retrieved May 29, 2024, from https://lifearchitect.substack.com/p/the-memo-special-edition-claude-3

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, *30*.

Wadhwani, S. (2024, April). *Top three llms compared: Gpt-4 turbo vs. claude 3 opus vs. gemini 1.5 pro*. Retrieved May 29, 2024, from https://www.spiceworks.com/tech/artificial-intelligence/articles/top-three-large-language-models-compared/

Wasserkrug, S., Boussioux, L., Hertog, D. D., Mirzazadeh, F., Birbil, I., Kurtz, J., & Maragno, D. (2024). From large language models and optimization to decision optimization copilot: A research manifesto. *arXiv preprint arXiv:2402.16269*. https://doi.org/10.48550/arxiv.2402.16269

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., Le, Q. V., Zhou, D., et al. (2022). Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, *35*, 24824–24837. https://proceedings.neurips.cc/paper_files/paper/2022/file/9d5609613524ecf4f15af0f7b31abca4-Paper-Conference.pdf

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

Zellers, R., Holtzman, A., Bisk, Y., Farhadi, A., & Choi, Y. (2019). Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*. https://doi.org/10.48550/arXiv.1905.07830

Zhao, Z., Wallace, E., Feng, S., Klein, D., & Singh, S. (2021, 18–24 Jul). Calibrate before use: Improving few-shot performance of language models. In M. Meila & T. Zhang (Eds.), *Proceedings of the 38th international conference on machine learning* (pp. 12697–12706, Vol. 139). PMLR. https://proceedings.mlr.press/v139/zhao21c.html

Appendix

Example Problems

LP Problem Description.[7] 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. (Poler et al., 2014)

IP Problem Description.[8] Your goal is to invest in several of 10 possible investment strategies in the most optimal way. The historic returns of those strategies are stored in the file "investments_data.csv". Each column represents one strategy and the rows are the past investment outcomes. There is no index and the values are separated by a ;. The costs for investing in a given investment is stored in a vector A, which has one value for each strategy in order. The values are: [80, 340, 410, 50, 180, 221, 15, 348, 191, 225] You can only invest once into an investment. Unfortunately due to other costs and inflation, your available budget at this time is uncertain. There are four possible budget scenarios with different probabilities: scenario 1 with 1000 euros and probability of 0.55, scenario 2 with 1100 euros and probability of 0.4, scenario 3 with 900 euros and probability of 0.04, scenario 4 with 1200 euros and probability of 0.01.The tolerable probability of exceeding the budget is 0.4.Please formulate a mean-variance mathematical model for this optimization problem, considering the past performance of

[7] For the reference mathematical formulation, code, and data, check: https://github.com/RiVuss/LLMsForOptimizationModelling/tree/main/Datasets/LP_1_Production-in-cosmetics-firm

[8] For the reference mathematical formulation, code, and data, check: https://github.com/RiVuss/LLMsForOptimizationModelling/tree/main/Datasets/IP_2_Choosing_Investment_Strategies

investment strategies and the uncertain budget. You can take 2 as the risk parameter r. (problem data: Kurtz, J., personal communication, April 21, 2023; problem logic as seen in Cornuejols & Tütüncü, 2006)

MIP Problem Description.[9] 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. (Pedroso et al., 2019)

NLP Problem Description.[10] 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 optimises its daily profits. (Poler et al., 2014)

Overview of Dataset Characteristics

Table 7.1 indicates the general characteristics of each optimization problem. A more detailed description of specific constraint types per each problem and other generic characteristics can be found in the shared GitHub[1] folder.

[9]For the reference mathematical formulation, code, and data, check: https://github.com/RiVuss/LLMsForOptimizationModelling/tree/main/Datasets/MIP_1_Facility_Location_Allocation

[10]For the reference mathematical formulation, code, and data, check: https://github.com/RiVuss/LLMsForOptimizationModelling/tree/main/Datasets/NL_1_Production_planning_in_a_drinks_firm

Table 7.1: Overview of the Characteristics of each Problem in the Dataset

		Parameters	Variables			Constraints	
	Domain	**Included**	**Continuous**	**Integer**	**Binary**	**Regular**	**Difficult**
LP1	Production Planning	X	X			X	
LP2	Production Planning		X				X
LP3	Production Planning	X	X			X	
LP4	Production Planning	X	X			X	
IP1	Assignment				X	X	
IP2	Portfolio Optimization				X		X
IP3	Assignment				X		X
IP4	Other	X		X	X		X
MIP1	Facility Location		X		X		X
MIP2	Production Planning		X	X	X		X
MIP3	Production Planning		X		X		X
MIP4	Network Flow	X	X		X		X
NLP1	Production Planning	X	X			X	
NLP2	Production Planning	X	X			X	
NLP3	Other	X	X		X		X
NLP4	Other		X				X

Difficult constraints include robust, logic, and time dynamic constraints. Problems with difficult constraints also have regular constraints, but the table shows the difficulty level of the constraints on the problem level. No classification of objective functions is included as they are too diverse to be meaningfully grouped.

Combined Results

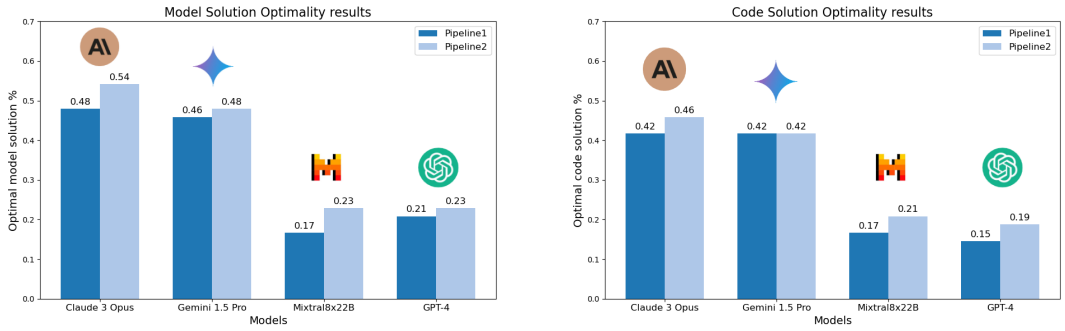
Aggregated Results

Table 7.2 compares the modeling and coding metrics aggregated over the whole dataset between the four LLMs for each pipeline.

Table 7.2: Comparison of Model and Code Metrics between the four LLMs for each Pipeline

	Pipeline 1 success % (count out of 48)				**Pipeline 2 success % (count out of 48)**			
	Claude 3 Opus	Gemini 1.5 Pro	Mixtral8x22B	GPT-4	Claude 3 Opus	Gemini 1.5 Pro	Mixtral8x22B	GPT-4
MM_1	41.67% (20)	37.5% (18)	2.08% (1)	6.25% (3)	45.83% (22)	45.83% (22)	14.58% (7)	8.33% (4)
MM_1^P	89.58% (43)	83.33% (40)	85.42% (41)	83.33% (40)	81.25% (39)	85.42% (41)	72.92% (35)	85.42% (41)
MM_1^V	83.33% (40)	81.25% (39)	37.5% (18)	50.0% (24)	89.58% (43)	95.83% (46)	54.17% (26)	54.17% (26)
MM_1^O	85.42% (41)	54.17% (26)	52.08% (25)	50.0% (24)	75.0% (36)	70.83% (34)	58.33% (28)	47.92% (23)
MM_1^C	43.75% (21)	50.0% (24)	8.33% (4)	10.42% (5)	50.0% (24)	58.33% (28)	22.92% (11)	16.67% (8)
MM_2	47.92% (23)	45.83% (22)	16.67% (8)	20.83% (10)	54.17% (26)	47.92% (23)	22.92% (11)	20.83% (10)
MC_1	93.75% (45)	79.17% (38)	83.33% (40)	60.42% (29)	85.42% (41)	93.75% (45)	83.33% (40)	58.33% (28)
MC_2	72.92% (35)	56.25% (27)	41.67% (20)	56.25% (27)	62.5% (30)	79.17% (38)	27.08% (13)	54.17% (26)
MC_3	41.67% (20)	41.67% (20)	16.67% (8)	14.58% (7)	45.83% (22)	41.67% (20)	20.83% (10)	16.67% (8)

Figure 7.1 shows a comparison between the different LLMs of the Model Objective Value Optimality (MM_2) and Code Objective value Optimality (MC_3) aggregated over the whole dataset.



(a) Model Objective Value Optimality (MM_2) (b) Code Objective Value Optimality (MC_3)

Figure 7.1: Model and Code Objective Value Optimality (MM_2 and MC_3) aggregated over the whole dataset. For MM_2, *Claude 3 Opus* performed best across both pipelines, followed by *Gemini 1.5 Pro*. *GPT-4* performed better than *Mixtral 8x22B* in *pipeline 1* and equally in *pipeline 2*. For MC_3, *Claude 3 Opus* performed best, followed by *Gemini 1.5 Pro*, *Mixtral 8x22B* and *GPT-4*. For both metrics, *pipeline 2* results are better than or, in one case, the same as in *pipeline 1*.

Figure 7.2 shows a comparison between the different LLMs of the Model Consistency (MC_3) aggregated over the whole dataset.

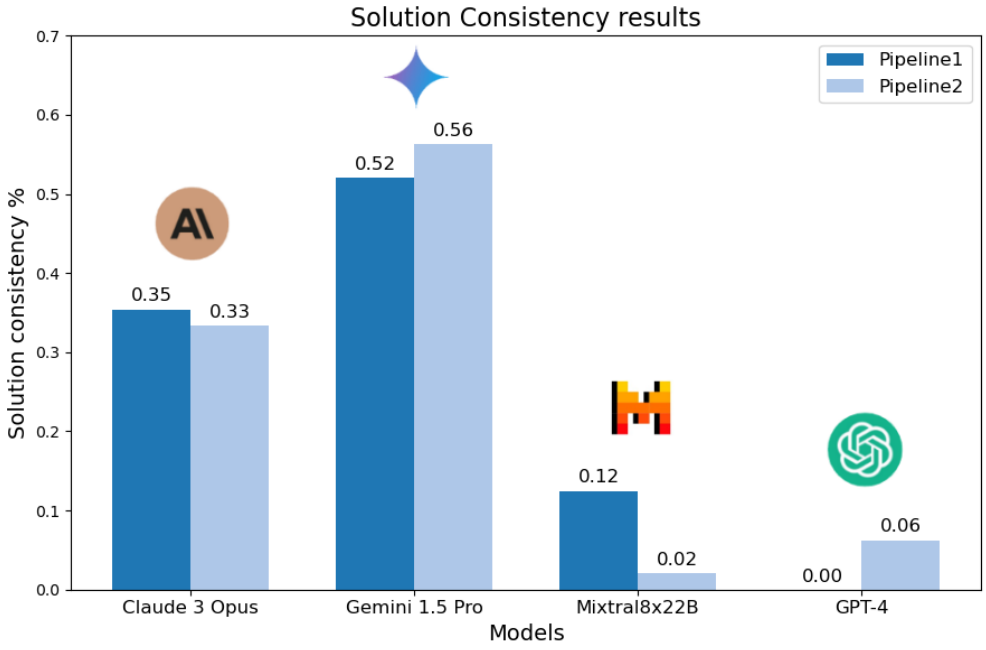


Figure 7.2: Model Consistency (MC_3) aggregated over the whole dataset. In *pipeline 1*, *Gemini 1.5 Pro* generated the most consistent models, followed by *Claude 3 Opus*, *Mixtral 8x22B* and *GPT-4*. In *pipeline 2*, *Gemini 1.5 Pro* generated the most consistent models, followed by *Claude 3 Opus*, *GPT-4* and *Mixtral 8x22B*. *Pipeline 1* was more consistent for *Claude 3 Opus* and *Mixtral 8x22B*, *pipeline 2* for *Gemini 1.5* and *GPT-4*.

Results per Trial

Figure 7.3 shows the Model Objective Value Optimality (MM_2) and Code Objective value Optimality (MC_3) per individual run across both pipelines.

(a) Pipeline 1 (b) Pipeline 2

Figure 7.3: Breakdown of Model and Code Objective Value Optimality (MM_2 and MC_3). Across both pipelines, there were four problems that were never solved correctly by any of the LLMs: IP4, MIP4, NLP3, and NLP4. Four problems were also solved correctly at least once in each pipeline by all LLMs: IP3, LP1, and NLP1. No problem was correctly solved in all three runs in both pipelines by all LLMs.

Statistical Significance of Results

Mathematical Model Evaluation

Table 7.3 shows the p-values obtained from performing several paired t-tests for the difference in mathematical model component correctness and model objective value optimality means between the two pipelines. The pairs tested were: $\mathrm{P}_{p1} \| \mathrm{P}_{p2}$, $\mathrm{V}_{p1} \| \mathrm{V}_{p2}$, $\mathrm{O}_{p1} \| \mathrm{O}_{p2}$, $\mathrm{C}_{p1} \| \mathrm{C}_{p2}$, , $\mathrm{Model}_{p1} \| \mathrm{Model}_{p2}$, and $\mathrm{Solution}_{p1} \| \mathrm{Solution}_{p2}$.

Table 7.3: P-Values for Differences of $MM_1^{Component}$ and MM_2 across the two Pipelines

	Parameters	Variables	Objective	Constraints	Model	Solution
p-value	0.1693	0.0825	0.2369	0.3828	0.6091	0.3332

The p-values for mathematical model component correctness and model objective value optimality compared across the two pipelines indicate whether there are significant differences between the pipelines. There is no statistically significant difference.

Table 7.4 shows the p-values obtained from performing several independent t-tests for the difference in the correctness of the mathematical components. The pairs tested were: P‖V, P‖O, P‖C, V‖O, , V‖C, and O‖C.

Table 7.4: P-Values for Differences of $MM_1^{component}$ across the Model Components

	P‖V	P‖O	P‖C	V‖O	V‖C	O‖C
Pipeline 1	0.569	0.7034	**0.0009***	0.8651	**0.0061***	**0.0042***
Pipeline 2	0.448	0.6165	**0.0404***	0.2288	**0.0090***	0.1142

The p-values for model correctness comparisons across different optimization model components (P, V, O, C). In both pipelines, there is a statistically significant difference between parameter and constraint correctness as well as variable and constraint correctness. In pipeline 1, the difference between objective and constraint correctness is also statistically significant.

Code Evaluation

Table 7.5 shows the p-values obtained from performing several paired t-tests for the difference in code metrics (MC_1, MC_2 and MC_3) means between the two pipelines. The pairs tested were: $MC_{1p1} \| MC_{1p2}$, $MC_{2p1} \| MC_{2p2}$, and $MC_{3p1} \| MC_{3p2}$.

Table 7.5: P-Values for Differences of MC_1, MC_2 and MC_3 across the two Pipelines

	Translation	Executability	Solution
p-value	0.2611	0.1359	0.4973

The p-values for code metrics compared across the two pipelines indicate whether there are significant differences between the pipelines. There is no statistically significant difference.

Consistency Evaluation

Table 7.6 shows the p-values obtained from performing a paired t-test for the difference in consistency (MM_3) means between the two pipelines.

Table 7.6: P-Values for Differences of MM_3 across the two Pipelines

	Consistency
p-value	0.8071

The p-values for consistency compared across the two pipelines indicate whether there is a significant difference between the pipelines. There is no statistically significant difference.

Impact of Problem Class on Performance

Table 7.7 shows the p-values obtained from performing several independent t-tests for the difference in model correctness means between the four optimization classes. The pairs tested for each pipeline were: IP‖LP, IP‖MIP, IP‖NLP, LP‖MIP, LP‖NLP, and MIP‖NLP.

Table 7.7: P-Values for Differences of MM_1 across the Optimization Classes

	IP‖LP	IP‖MIP	IP‖NLP	LP‖MIP	LP‖NLP	MIP‖NLP
Pipeline 1	0.8090	0.4943	0.3822	0.6202	0.4772	0.8164
Pipeline 2	0.5370	1.0000	0.2148	0.5370	**0.0447***	0.2148

The p-values for model correctness comparisons across different optimization classes (IP, LP, MIP, NLP). Only in *pipeline 2*, there is a significant difference in *Claude 3 Opus* ability to model problems of classes LP and NLP.

Table 7.8 shows the p-values obtained from performing several paired t-tests for the difference in model correctness means between the two pipelines for each optimization class. The pairs tested were: $\text{IP}_{p1}\|\text{IP}_{p2}$, $\text{LP}_{p1}\|\text{LP}_{p2}$, $\text{MIP}_{p1}\|\text{MIP}_{p2}$, and $\text{NLP}_{p1}\|\text{NLP}_{p2}$.

Table 7.8: P-Values for Differences of MM_1 per Optimization Class across the two Pipelines

	IP	LP	MIP	NLP
p-value	0.3910	0.2152	0.3910	0.3910

The p-values for model correctness per optimization class (IP, LP, MIP, NLP) compared across the two pipelines indicate whether there are significant differences between the pipelines. There is no statistically significant difference.

Table 7.9 shows the p-values obtained from performing several independent t-tests for the difference in code objective value optimality means between the four optimization classes. The pairs tested for each pipeline were: IP‖LP, IP‖MIP, IP‖NLP, LP‖MIP, LP‖NLP, and MIP‖NLP.

Table 7.9: P-Values for Differences of MC_3 across the Optimization Classes

	IP‖LP	IP‖MIP	IP‖NLP	LP‖MIP	LP‖NLP	MIP‖NLP
Pipeline 1	1.0000	0.3822	0.3822	0.3464	0.3464	1.0000
Pipeline 2	0.1066	0.6540	0.6540	**0.0447***	**0.0447***	1.0000

The p-values for code objective value optimality comparisons across different optimization classes (IP, LP, MIP, NLP). Only in *pipeline 2*, there is a significant difference in *Claude 3 Opus* ability to solve problems of classes LP and MIP as well as LP and NLP using Pyomo in Python.

Table 7.10 shows the p-values obtained from performing several paired t-tests for the difference in objective value optimality means between the two pipelines for each optimization class. The pairs tested were: $\text{IP}_{p1}\|\text{IP}_{p2}$, $\text{LP}_{p1}\|\text{LP}_{p2}$, $\text{MIP}_{p1}\|\text{MIP}_{p2}$, and $\text{NLP}_{p1}\|\text{NLP}_{p2}$.

Table 7.10: P-Values for Differences of MC_3 per Optimization Class across the two Pipelines

	IP	LP	MIP	NLP
p-value	0.1817	0.0917	1.0000	1.000

The p-values for code objective value optimality per optimization class (IP, LP, MIP, NLP) compared across the two pipelines indicate whether there are significant differences between the pipelines. There is no statistically significant difference.

Table 7.11 shows the p-values obtained from performing several independent t-tests for the difference in the frequency of executability mistakes between the four optimization classes. The pairs tested for each pipeline were: IP‖LP, IP‖MIP, IP‖NLP, LP‖MIP, LP‖NLP, and MIP‖NLP.

Table 7.11: P-Values for Differences of Frequency of Executability Mistakes across the Optimization Classes

	IP‖LP	IP‖MIP	IP‖NLP	LP‖MIP	LP‖NLP	MIP‖NLP
Pipeline 1	1.0000	0.3559	1.0000	0.3559	1.0000	0.3559
Pipeline 2	0.3559	0.5504	0.3559	0.3559	1.0000	0.3559

The p-values for frequency of executability mistakes comparisons across different optimization classes (IP, LP, MIP, NLP). There is no statistically significant difference.

Table 7.12 shows the p-values obtained from performing several independent t-tests for the difference in the frequency of "harmless" modeling mistakes between the four optimization classes. The pairs tested for each pipeline were: IP‖LP, IP‖MIP, IP‖NLP, LP‖MIP, LP‖NLP, and MIP‖NLP.

Table 7.12: P-Values for Differences of Frequency of Harmless Modeling Mistakes across the Optimization Classes

	IP‖LP	IP‖MIP	IP‖NLP	LP‖MIP	LP‖NLP	MIP‖NLP
Pipeline 1	0.3559	1.0000	1.0000	0.3559	0.3559	1.0000
Pipeline 2	0.3559	1.0000	0.3559	0.3559	1.0000	0.3559

The p-values for the frequency of "harmless" modeling mistakes comparisons across different optimization classes (IP, LP, MIP, NLP). There is no statistically significant difference.

Impact of Constraint Complexity on Performance

Table 7.13 shows the p-values obtained from performing an independent t-test for the difference in constraint correctness between the two constraint complexity levels, regular and difficult.

Table 7.13: P-Values for Differences of Constraint Correctness (MM_1^C) across the Complexity Levels

	Regular‖Difficult
Pipeline 1	0.1180
Pipeline 2	0.1643

The p-values for the constraint correctness comparison across different constraint complexity levels (regular, difficult). There is no statistically significant difference.