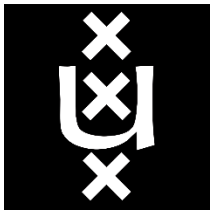
Thesis title:

Large Language Models for Optimization Modelling:

Analyzing the performance of Gemini 1.5 Pro

Author:
Hubert Perliński
13985612



University of Amsterdam
BSc Business Analytics

UvA supervisor:
Donato Maragno

26.06.2024

Abstract

While optimization modelling has been shown to lead to significant societal and financial benefits in various fields, it remains underutilized due to the expert knowledge needed for its application. Early research demonstrated that Large Language Models (LLMs) could potentially allow business experts to use optimization without requiring advanced mathematical knowledge. However, before this vision becomes a reality, the performance of contemporary LLMs in modelling and solving realistic business problems needs to be better understood, which is the focus of this study. To this end, a dataset of 16 diverse business optimization problems was curated, encompassing four different optimization classes and varying problem difficulty. It was experimented on using the Gemini 1.5 Pro model across a one-step pipeline and a chain-of-thought pipeline, which decomposes the problem into smaller sub-tasks. Key performance metrics were employed to evaluate the model outputs. The results suggest that while Gemini 1.5 Pro can solve some optimization problems, finding the optimal solution 41.7% of the time in both pipelines, there still are notable failure patterns that would need to be addressed to use the model autonomously in real life. Its outputs were also not consistent across multiple runs, highlighting the need for further refinement.

Contents

1 Introduction

Optimization is a collection of mathematical methods used for solving quantitative problems (Wright, 2024). It allows the identification of the best possible solution among a set of feasible options, given an objective function to minimize or maximize and a set of constraints to satisfy. Modelling real-world situations as optimization problems (also known as mathematical programming) can lead to significant benefits across different industries. Among others, it has allowed Kim and Reed (2010) to improve the production of useful chemicals in bacteria like E. coli, by optimizing genetic modifications. Benalcazar et al. (2024) have shown that operating costs of energy-intensive companies can be lowered by up to 20% with electricity system optimization. The financial gains brought by mathematical programming can also have important societal benefits, such as in the case study by den Hertog et al. (2019), in which the authors used optimization to decrease the operational costs of The World Food Programme's mission in Iraq by 12%, while not compromising the nutritional value supplied.

However, in most projects, business experts are still not utilizing mathematical programming to its full potential, as it remains predominantly within the domain of optimization experts (Schulz, 2021). This may be due to a few reasons. The decision-makers may lack the mathematical skills necessary to formulate problems effectively and may not even be aware that certain aspects of their operations can be optimized. If they recognize an opportunity, they may lack access to professionals with the expertise required or may be hesitant to invest in their work due to the uncertainty surrounding its outcome. Those factors contribute to the underutilization of optimization modelling, however, new technologies may alleviate their impact. One such advancement are Large Language Models (LLMs), which have been the focus of a rapidly increasing amount of research (Zhao et al., 2023). Chat interfaces using LLMs, such as ChatGPT (OpenAI, 2022) have widely popularized their use in many tasks including information retrieval, text, and code generation. With the increased interest in the technology more companies started releasing ever better LLMs, such as the Llama model family from Meta (Touvron et al., 2023) and the Gemini model family from Google (Google et al., 2023). Advanced contemporary models have been used as personal co-pilots created for a specific purpose, such as math tutors, tech advisors, and coding assistants (OpenAI, 2023b; Hsiao, 2024).

Wasserkrug et al. (2024) have outlined a vision for combining optimization with LLMs, thus creating an AI co-pilot with the purpose of solving problem-specific optimization models, based on their natural language descriptions. Such a tool would allow decision-makers to simply explain their situation to the bot and be provided with the best option they can choose, making optimization widely accessible. Bridging the divide between business experts and mathematical programming professionals could lead to immense financial and societal benefits, which is why researchers have started exploring the capabilities of LLMs in this field,

with early research showing promising results. OptiMUS agent developed by AhmadiTeshnizi et al. (2023) can solve up to 60% of optimization problems following a strict input schema, proposed by the authors, with no human intervention. It not only uses the LLM to solve the problems, but also to debug the output if any problems arise. Ahmed and Choudhury (2024) achieved similar results using the GPT-4 model (OpenAI, 2023a), which reached a 63% F1-score on a dataset of linear optimization problems. Their research also highlighted a significant gap in the contextual understanding capabilities of smaller LLMs, namely Llama-2-7b (Touvron et al., 2023), compared to larger counterparts. Li et al. (2023) have shown that using a more advanced pipeline for the task of optimization model generation can lead to substantial improvements in performance as compared to one-step pipelines. While these findings are impressive, they highlight just a fraction of what is to be explored in this emerging topic of LLM application for optimization modelling. This study aims to contribute to the literature by looking at the performance of the newest models and testing them on more varied datasets.

The LLM examined in this report was Google's Gemini 1.5 Pro, which represents a significant advancement in the technology (Google, 2024). Building on the company's extensive experience, including the creation of the Transformer architecture (Vaswani et al., 2017), BERT (Devlin et al., 2018), PaLM (Chowdhery et al., 2022; Anil et al., 2023) and Gemini 1.0 (Google et al., 2023) models, Gemini 1.5 Pro incorporates a mixture-of-experts architecture, which dynamically selects a subset of model components to optimize performance across tasks (Shazeer et al., 2017). It can also process up to 10 million tokens, a substantial increase from previous state-of-the-art models. Notably, despite using fewer parameters than its predecessor, Gemini 1.0 Ultra, it matches or surpasses it across various benchmarks, including math, science, and reasoning, which are critical for optimization modelling. Additionally, the model supports multiple input modes, such as text, audio, images, and video, advancing the state-of-the-art in multimodal tasks. However, its potential in mathematical programming remains to be evaluated, as the model was only publicly released in May 2024.

The next sections of this report are organized as follows: first, the current state of the relevant research is described in the literature review. Then the contributions of this study are outlined. Next, the exact methodology of the experiment is discussed, including the process of gathering the data, the generation pipeline designs, and the metrics gathered. After that, the results of the research are presented. Lastly, the limitations of the study are considered, and conclusions are drawn.

2 Literature review

Although optimization modelling and large language models have been researched for a while separately, the field of applying LLMs in mathematical programming tasks has emerged recently. A notable example of early exploration of this task is described in the paper by Ramamonjison et al. (2023) about the Natural Language for Optimization (NL4Opt) competition. The goal of the contest, which took place in 2022, was to increase the accessibility of problem solvers by creating a method for non-experts to interface with them using natural language. The process was split into two sub-tasks: labeling semantic entities that correspond to components of optimization problems in their descriptions and generating a logical form representation of the problem, which could be input into a solver. Due to the limitations of contemporary LLMs, the first step was necessary for achieving good results. For the contest, the organizers prepared a dataset of 1101 annotated simple linear programming (LP) problems, which to this day is the largest of its kind and has contributed to other literature (Li et al., 2023; Ahmed & Choudhury, 2024). The winning team of the second sub-task managed to achieve a solution accuracy of 89.9% by enriching the input by surrounding entities with XML-like tagging (Gangwar & Kani, 2022). The top 3 teams in the competition all used Facebook's BART language models (Lewis et al., 2019). While the contest's winners were announced on the 4th of November 2022, ChatGPT was publicly released on the 30th of November (OpenAI, 2022), hence Ramamonjison et al. have decided to include it as a comparison in their report on the competition. With no intermediate entity tagging, it managed to achieve an accuracy of 92.7%, beating all contest entries. Its most common modes of failure were incorrect variable coefficients, extra variables, extra or missing constraints, and wrong constraint directions. Although this paper has demonstrated the viability of using LLMs for optimization modelling, its major limitation was that all the problems in the dataset were simple linear programming tasks, while the real-world applications are much more complex.

An example of a paper that benefited from using the NL4Opt dataset is the work of Ahmed and Choudhury (2024), who have compared the performance of fine-tuned Llama 2-7B to the results of GPT-3.5 and GPT-4 models. The fine-tuning was done using the NL4Opt data and the GSM8K dataset of math problems. Their research showed a large gap in the contextual understanding capabilities of smaller LLMs (such as Llama 2-7B). Even though the F1 score of Llama 2-7B was much smaller than that of GPT models (0.126 vs 0.493 for GPT-3.5 and 0.633 for GPT-4), the authors showed that fine-tuning LLMs could be a viable strategy for improving the performance of optimization tasks. Another contribution of Ahmed and Choudhury was comparing results obtained using zero-shot and 1-shot testing (where the number represents how many examples of correct solutions were provided in the prompt). This was proven to have a positive impact on the outcome in their case, however, it is important to note that since their test set contained only linear

programming problems, it is not known if the results would still be improved by this technique with a dataset consisting of different optimization domains.

Li et al. (2023) acknowledged the problem difficulty limitation present in the aforementioned papers and extended the NL4Opt dataset with more problem descriptions, which contained logic constraints and binary variables, making some problems belong to the mixed integer linear category (MILP). For their pipeline, they proposed a framework, which splits the task into three steps: decision variable identification, classification of the objective and constraints, and the generation of the mathematical models. Interestingly, for the second step, the authors fine-tuned an LLM to act as a classifier for 13 types of constraints that were in their dataset (with 7 appearing in the NL4Opt data). After the type of the constraint was identified, it could be formulated using a template. This experiment design was run using ChatGPT powered by GPT-3.5 and Google's Bard powered by the PaLM2 model (Anil et al., 2023). The results showed that in both cases the multi-stage approach greatly outperformed simply asking the LLM to generate the model. However, the framework relies on the assumption that the constraint classifier knows all the possible constraint types, which may not be realistic if the problem is not known beforehand. It was not assessed on, for example, non-linear constraints.

Another pipeline, called OptiMUS, was proposed by AhmadiTeshnizi et al. (2023). It is agnostic to constraint types, but relies on a problem description structure called SNOP (Structured Natural Language Optimization Problem), proposed by the authors. For their experiment, they collected a dataset of 52 LP and MILP problems following that format. The solution includes several stages. First, the mathematical problem formulation is generated based on the SNOP description. Then, its Python code representation is generated, using the Gurobi solver and the cvxpy library. Next is the testing and revision stage, where if running the code the LLM produced fails, the error message is added to the prompt and the LLM is asked to fix it. This process repeats until the code is executable or the maximum number of iterations is reached. Then, the LLM is tasked with creating tests to verify that the calculated solution is viable. If the tests fail, a similar cycle of revisions takes place. Finally, as an additional strategy, the LLM is asked to rephrase the problem description and run the previous steps again. Using this pipeline, the authors achieved a 60% success rate on their test data, solving nearly twice as many problems compared to a basic LLM prompting strategy.

A different approach was taken by Amarasinghe et al. (2023), who focused on fine-tuning an LLM to work on problems from one domain. They have constructed a dataset of 100 problem descriptions and formulations specifically in the field of production scheduling. The model chosen for their pipeline was CodeRL, a relatively small LLM focused on generating executable code (Le et al., 2022). Due to the token limitations of this LLM, the authors had to use modularization to split the task into smaller bits and generate the formulation in a few parts. This approach produced very good results, proving that fine-tuning may be

a sufficient strategy if the optimization problems are from the same field. The authors also highlight the importance of incorporating explanation techniques to help practitioners validate the generated results and prevent consequences of the copilot providing wrong solutions.

On the other hand, Yang et al. (2023) considered a heuristic technique for solving complex optimization problems with LLMs, which they called Optimization by PROmpting (OPRO). It is based on an iterative process. Given the problem description, the LLM is asked right away for the solution. Then its answer is evaluated against the objective. After, the solution is added to the prompt with its result and the process repeats. While not precise, when this approach was used with GPT-4, it worked surprisingly well as a heuristic. The produced solutions were of good quality and sometimes matched or surpassed hand-designed heuristic algorithms, making OPRO potentially useful for computationally hard problems. Those results could not however be replicated with less advanced LLMs. The authors demonstrated that this technique worked for the travelling salesman problem, however, it could prove the most useful in tasks including a black-box function, such as LLM prompting. Yang et al. applied it to optimize the prompt for an agent solving math problems and have achieved significantly better results than the ones previously described in the literature.

Even though the early research shows that there is potential in applying LLMs to optimization modelling, there are still many areas requiring more thorough examination, such as considering more difficult problems of different optimization classes. The capabilities of new models should also be tested and compared to understand what makes an LLM good for this application.

3 Contributions

The field of LLMs for optimization research is rapidly evolving with frequent releases of new models. This report is part of a project that aims to contribute to the existing literature by looking at the performance of cutting-edge models, which due to their recency, have not yet been exhaustively analyzed. It is to provide insight into the state-of-the-art and the difficulties that remain in making an optimization co-pilot viable. Hence, the main research question is **"How capable are contemporary LLMs in designing and solving optimization models, starting from natural language descriptions of problems with varying complexity?"**.

The models investigated in the project were Claude 3 Opus (Anthropic, 2024), Gemini 1.5 Pro (Google, 2024), GPT-4 (OpenAI, 2023a), and Mixtral 8x22b (Mistral AI, 2024). This report focuses exclusively on the performance of Gemini 1.5 Pro, which was publicly released on the 15th of May 2024, however, the results comparison to the other models can be found in the Appendix.

To answer the research question, further sub-questions have been formulated, an overview of which can be found in Figure 1. Wasserkrug et al. (2024) note the ability to translate business problem definitions into mathematical models as a crucial requirement for an optimization co-pilot, which is why the capabilities of LLMs in that task were the first interest of the study. To execute this research goal, a set of 16 diverse natural language descriptions of optimization problems and their solutions was gathered. Then the LLM was used to generate mathematical models based on problem descriptions. In contrast to AhmadiTeshnizi et al. (2023), it was decided that the problem descriptions should not follow any input structure. In that way, they resemble more accurately how a businessperson could describe a problem. Although fewer problems than in other studies were used, they were the most varied across difficulty and optimization classes included (linear, integer, mixed-integer, and non-linear). This allowed to identify failure patterns, which would not be otherwise apparent and see if performance varies across classes.

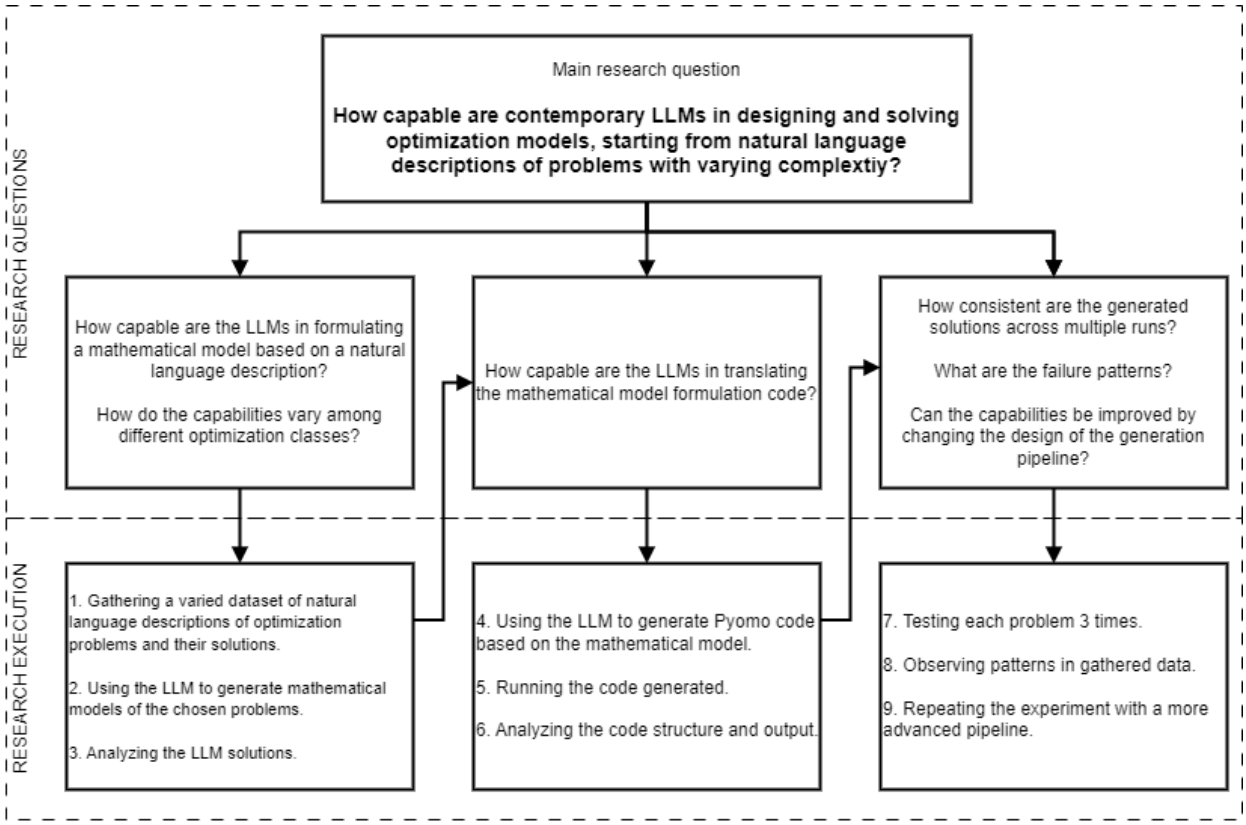


Figure 1: Overview of the research questions and the execution plan of the study.

Secondly, for an optimization co-pilot to provide the optimal solution to a problem, it needs to communicate its formulation to a solver. Such ability was analyzed by asking the LLM to generate code using the Python Pyomo optimization library, based on the mathematical formulation it has come up with in the previous step. This code was then executed, and the calculated solution was compared to the correct one. This builds upon

the experiment design of Li et al. (2023), who have only looked at the ability to generate the mathematical formulation and not the code required to solve it.

The third interest of the study was the analysis of the consistency and failure patterns of the solutions generated by the LLM. Wasserkrug et al. (2024) highlight that the outputs generated by the models are not deterministic and may be correct only in some runs. Other literature does not consider that possibility, so in this experiment each problem was run 3 separate times, allowing to see if the LLM's answers were stable. Throughout these iterations, various metrics and comments were collected to observe any emerging patterns. Furthermore, it was sought to see if, like in Li et al. (2023), a more advanced generation pipeline design could prevent the aforementioned failure modes. To check that, the experiment was repeated with a second pipeline using a chain-of-thought approach (Besta et al., 2024).

4 Methodology

4.1 Data description

The dataset used to conduct the experiments has been curated with the goal of assessing the LLMs' 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 (Ahmed & Choudhury, 2024; Xiao et al., 2024), but as mentioned by Ramamonjison et al. and Amarasinghe et al. (2023), its shortcoming is that it consists of only LP problems, while many real-world use cases require the introduction of integer variables or even include nonlinear terms. The dataset used by Li et al. (2023) builds upon the NL4OPT and introduces some logic constraints as well as integer variables to create a set of more complex problems. The dataset used by AhmadiTeshnizi et al. (2023) consists of LP and MIP problems following the SNOP structure. Amarasinghe et al. (2023) investigate exclusively production 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 performs on various kinds of real-world optimization problems characterized by their diversity in terms of optimization classes, difficulty levels, and domains.

Considering the characteristics and shortcomings of the above-mentioned datasets, the data used in this thesis consists of a more diverse set of mathematical optimization problems. Given the time span and scope of this thesis, a total of 16 problems was selected. For each of them, the data consists of a natural language problem description, its mathematical formulation, the Python Pyomo code calculating the optimal solution, and any necessary data files containing parameter values. While the natural language description serves as

input for the LLM, the mathematical formulations and respective Pyomo code serve evaluation purposes only.

Problems span four types of optimization classes: linear programming (LP), integer programming (IP), mixed integer programming (MIP), and nonlinear programming (NLP). For each of the four optimization classes, four problems of varying difficulty and business domains were selected. More complex problems are mainly characterized by their use of non-trivial constraint types such as logic constraints or the need for unit conversion of decision variables. The use of such problems was inspired by Li et al. (2023), who introduced more complexity into the dataset used in the NL4OPT competition by adding four types of logic constraints. They also remarked that the LLMs used in their study struggled with unit conversions. Due to the nature of the use cases, some of the problem descriptions are formulated explicitly including the parameter values while others, especially larger problems, implicitly define the problem without supplying any 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 different 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 and mathematical formulation can be found in Appendix B and a summary of the characteristics of all problems in Appendix C.

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, n.d. Pedroso et al., n.d. StemEZ, n.d.). Gemini 1.5 Pro's training data includes information up to November 2023 (Google, n.d.). As most of the data sources used were readily available online at that time, 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 variables were added, the objective function or constraints were modified or some parameter values were changed. For a majority of the problems, no Python Pyomo code was available, hence it was written by the members of the project group themselves. Both of these factors help mitigate the risk of using optimization problems as part of this research that the LLM has been trained over. Overall, using a set of such diverse optimization problems aids in more realistically assessing whether Gemini 1.5 Pro could be suitable for applications in real business settings, for example, the Decision Optimization CoPilot as envisioned by Wasserkrug et al. (2024).

4.2 Experiment 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 et al., 2023). To assess if a more advanced generation pipeline can improve the LLM's capabilities, the problems gathered for this study were experimented on across two pipelines of different complexities. Both approaches were designed to generate two main outputs to be evaluated: the **mathematical model** of the given problem and its Python **Pyomo code** representation. Those outputs were later saved to a repository and assessed by the metrics described further in the document. Pyomo is an open-source Python library for formulating, solving, and analyzing optimization models (Hart et al., 2011). It was chosen for its support of a wide range of problem types and our familiarity with its syntax.

4.2.1 Pipeline 1

The idea for the first pipeline was to emulate a simple one-step LLM query, in which the user asks to model their problem and attaches the problem description. A similar approach was taken by Fan et al. (2024) in their experiment. This is also how a business user would likely interact with a contemporary LLM chat interface. The problem description was left unaltered and appended to the prompt, similar to the methodology outlined in Ramamonjison 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 the Gemini 1.5 Pro's input and output in this step can be seen in Figure 2.

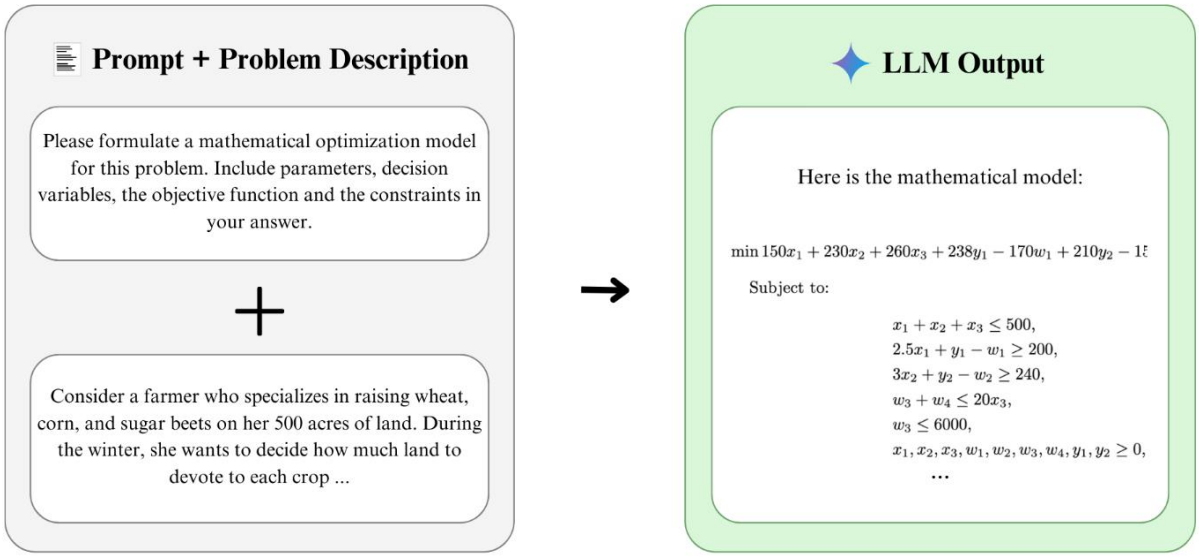


Figure 2: The modelling step of the experiment Pipeline 1.

Next, with the first LLM response as input, the LLM was asked to generate the Pyomo code corresponding to the mathematical formulation with the following prompt (as seen in Figure 3):

"Please write Pyomo code for this mathematical problem. Use sample data where needed. Indicate where you use sample data." + previous LLM 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. For the problems requiring data, it was loaded manually, according to the data structure assumed by the LLM. Thanks to this, the problem descriptions could be more realistic, without precise data format descriptions. At this stage, it was also checked if the LLM correctly converted the mathematical model into code. In cases when this step failed, an extra piece of code was written manually to verify the viability of the generated mathematical formulation.

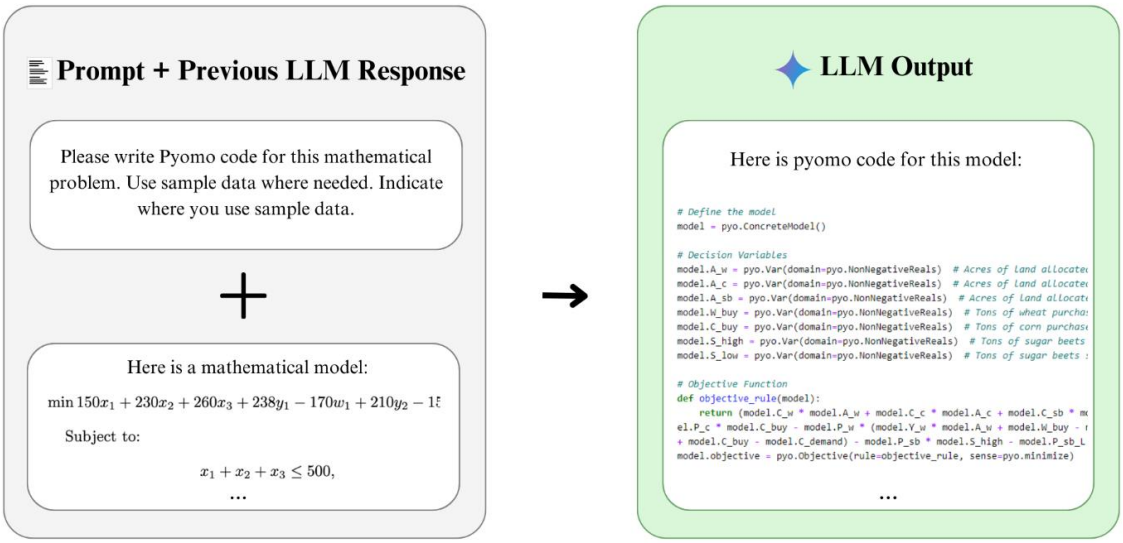


Figure 3: The coding step of the experiment Pipeline 2.

4.2.2 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 3 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 variables, then the objective function, and lastly the constraints, with the following prompts:

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

Such division of work results in smaller subtasks which should be easier for the LLM to solve, potentially leading to an improved final outcome (Best et al., 2024). The three responses were then added together to form the full mathematical model. This approach can be also seen in Figure 4.

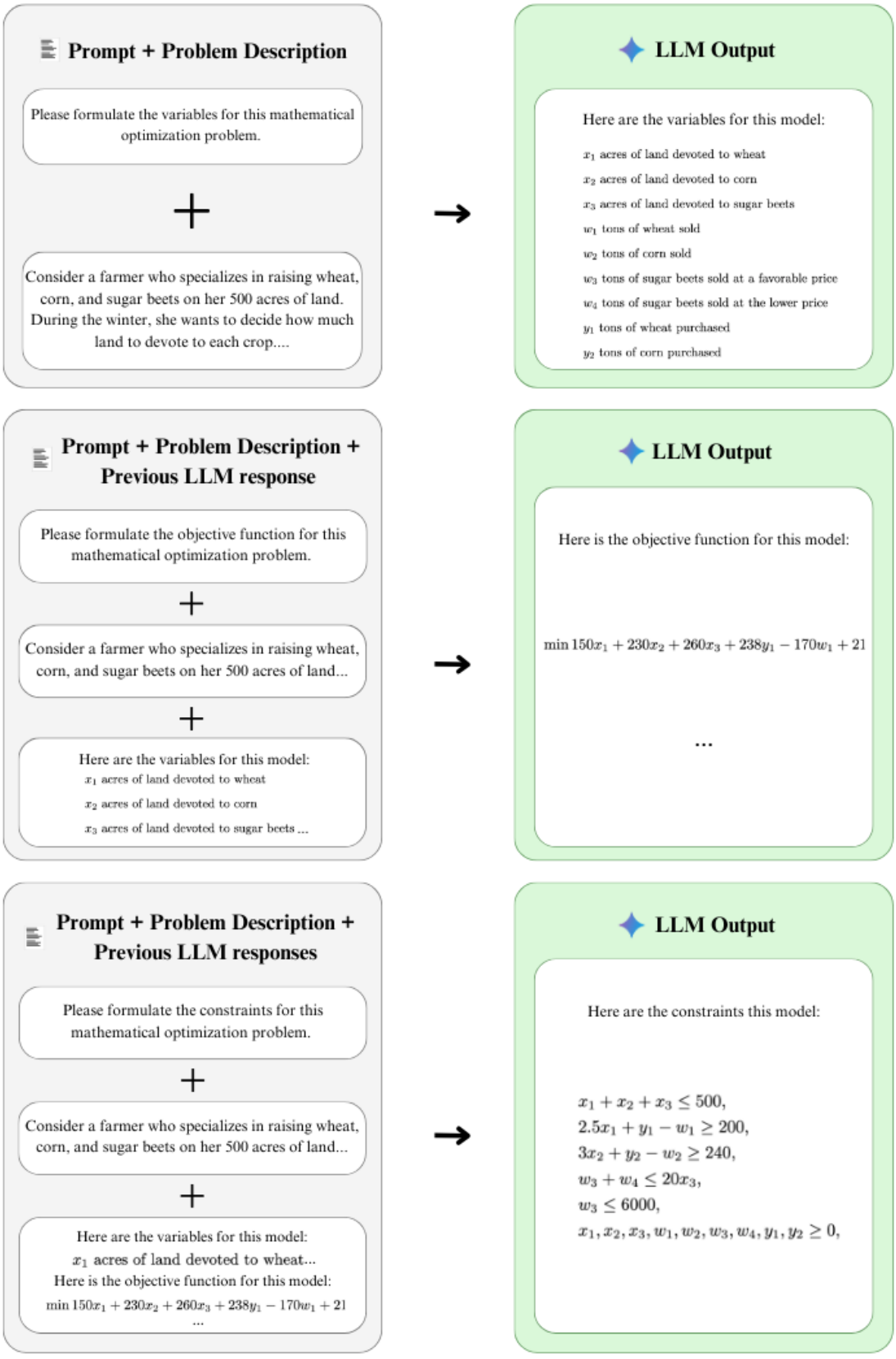


Figure 4: The three steps of the modelling part of experiment Pipeline 2.

After the steps were combined to form the full model, the LLM was asked to generate Pyomo code for the model in the same way as in Pipeline 1, as seen in Figure 5. An overview of the pipelines is also shown in Figure 6. Next, the generated outputs were evaluated with the chosen metrics.

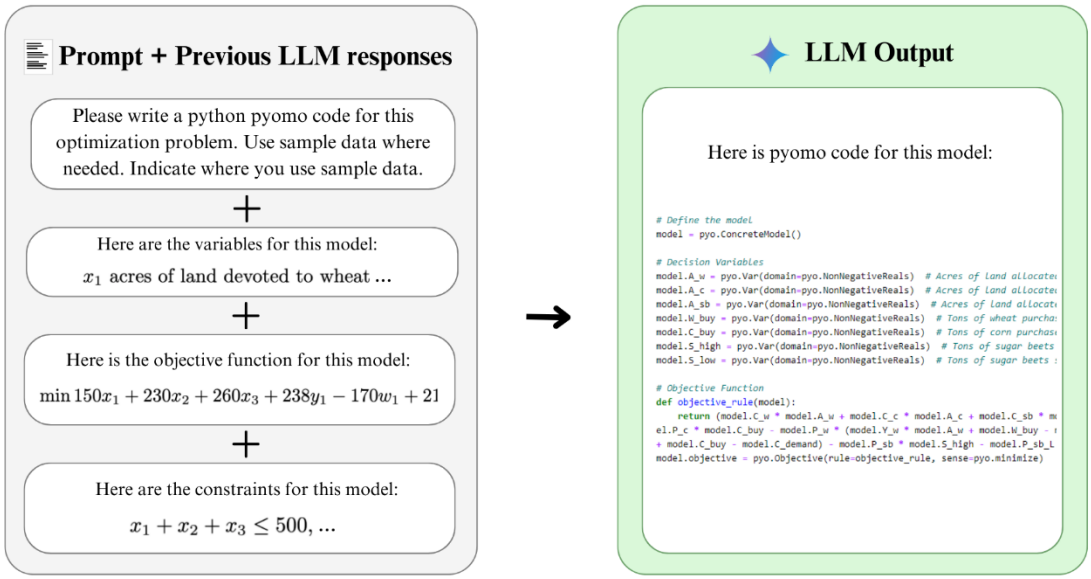


Figure 5: The coding step of the experiment Pipeline 2.

4.3 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 as an evaluation of the generated mathematical model and an evaluation of the code to solve the generated mathematical model. All metrics were recorded manually. Sarker et al. (2024) analyzed the ability of LLMs to produce code that solves mathematical equations and defined several key concepts relevant to the metrics used in this experiment. One such concept is the equivalence of semantics between two mathematical models, which means the models produce the same outputs for all possible inputs. Another definition is the syntax of a mathematical problem meaning the sequence of characters describing it. Changing the syntax of a mathematical problem while preserving its semantics is called a syntactic transformation. So, the problem description, reference mathematical formulation, and reference code for a given problem are semantically equal but syntactically different. The metric measurements focused on analyzing and comparing the semantics of LLM outputs and reference solutions. Thus, if, for example a constraint is formulated differently from the reference, it would still be considered correct as long as the semantics match the reference.

4.3.1 Correctness of Mathematical Model (MM1)

This metric represents the LLM's ability to produce a fully correct mathematical representation of the underlying problem as described in Li et al. (2023) and Ramamonjison 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, variables, the objective function, and constraints of the solution generated by LLM and the reference solution were measured. The equivalence is measured as a binary metric:

MM1P(M, R) = 1, if the parameters of the mathematical model (M) semantically match the parameters of the reference solution (R), 0 otherwise.

MM1V(M, R) = 1, if the variables of the mathematical model (M) semantically match the variables of the reference solution (R), 0 otherwise.

MM1O(M, R) = 1, if the objective function of the mathematical model (M) semantically matches the objective function of the reference solution (R), 0 otherwise.

MM1C(M, R) = 1, if the constraints of the mathematical model (M) semantically match the constraints of the reference solution (R), 0 otherwise.

MM1(M, R) = 1, if MM1P(M, R), MM1V(M, R), MM1O(M, R), MM1C(M, R) =1, 0 otherwise.

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 could possibly change parameters, variables, objective function, or constraints at any of the steps, thus leading to ambiguous conclusions.

4.3.2 Model Solution Optimality (MM2)

This metric measures if the objective value produced by the mathematical model of the LLM was optimal, considering the case where the generated mathematical model had additional inactive constraints, different variable domains, or other deviations from the reference model. Depending on the problem instance, these changes could still lead to a feasible solution with the optimal objective value. However, because the semantics of the formulations would be different, this optimal solution would not be captured by MM1. Thus, we define the Model Solution Optimality as

MM2(M, R) = 1, if the objective value of the generated mathematical model (M) and Reference solution (R) were equal, 0 otherwise.

4.3.3 Solution Consistency (MM3)

In addition to the accuracy of the generated mathematical model, we were also interested in the consistency of the outputs. To measure the consistency of LLMs, the outputs were paired as a Cartesian product and the number of semantically agreeing output pairs was divided by the total number of output pairs (Raj et al., 2023). Thus, we achieve the solution consistency metrics through:

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

where n = 3 is the number runs executed for a given problem, M is the i^{th} generated mathematical model to a given problem, and $I(M_i = M_j)$ is an indicator function for the semantic equivalence.

4.3.4 Equivalent Code Representation of Mathematical Model (MC1)

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

MM2(M, C) = 1, if the semantic meaning of the generated mathematical model (M1) and the conversion into code (C) were agreeing, 0, otherwise.

4.3.5 Executability of Model-Related Code (MC2)

The code executability metric measures how well the LLM was able to use the chosen coding tools (Python and Pyomo) to arrive at a solution after a mathematical formulation was provided. As was proposed by AhmadiTeshnizi et al. (2023) and Xiao et al. (2024) the generated code was tested for its executability, regardless of the solution value. Notably, the metric considers only the code related to the model definition and solving. Any mistakes in the parts of the code defining sample data or displaying the results were not counted, as they are not related to the goal of the experiment to test the LLM's modelling capabilities. Hence:

MC2(C) = 1, if the model-related code executed without errors, 0 otherwise.

4.3.6 Code Solution Optimality (MC3)

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 associated with the problem. The formula of this metric is:

MC3(C) = 1, if the objective value produced by the code generated by the LLM was correct, 0 otherwise.

By thoroughly combining these metrics in the evaluation, a precise description of the LLM's ability to solve optimization problems was achieved. It is important to note that all metrics except MM3 were measured for each solving attempt and later aggregated as seen in the results section. MM3 was measured once for each problem. An overview of the pipelines and the evaluation process can be found in Figure 6.

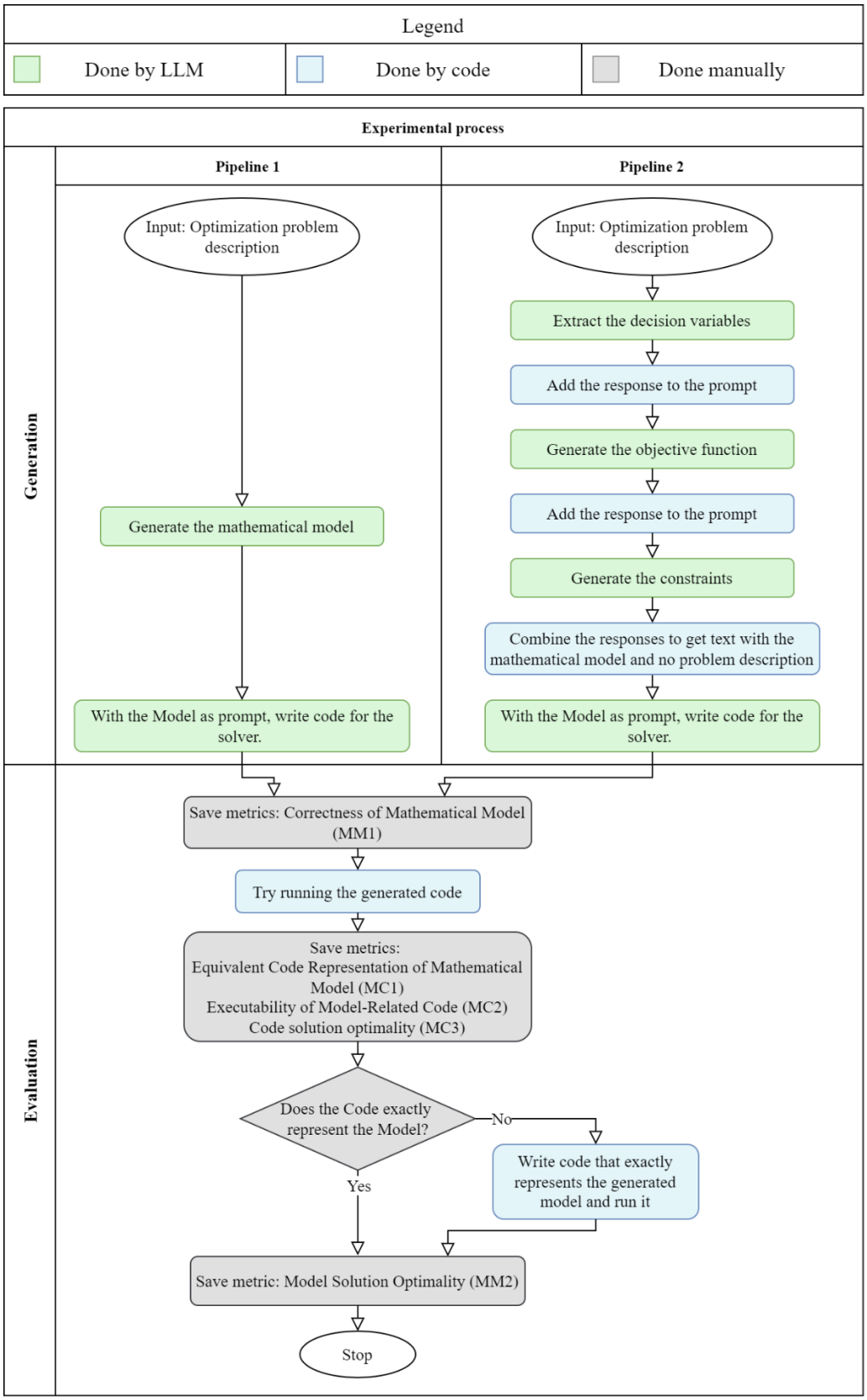


Figure 6: Overview of the experiment flow.

5 Results

The 16 problems in the dataset were tested 3 times each across two experiment pipelines, resulting in a total of 96 runs. The Model and Code Solution Optimality (MM2 and MC3) outcomes of each run can be seen in Figure 7. Other metrics have been aggregated in Figures 8, 9, and 10.

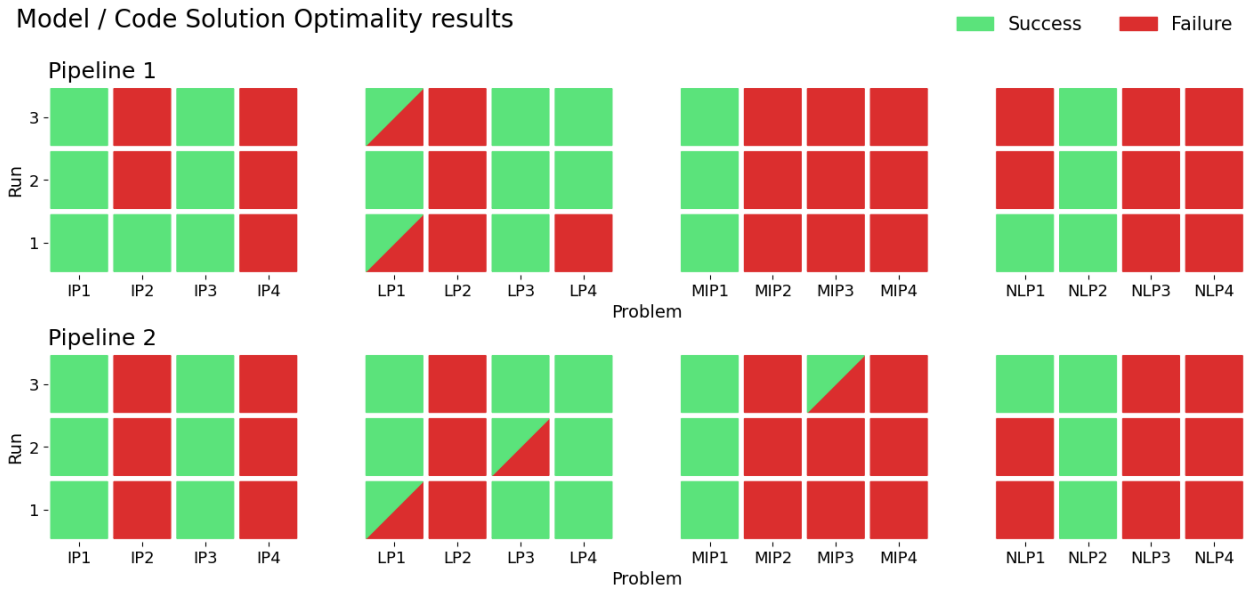


Figure 7: The optimality results of each run of the experiment, where the top left triangle indicates the model solution optimality and the bottom right triangle indicates the code solution optimality.

Metric	Pipeline 1 percentage (count)		Pipeline 2 percentage (count)	
	All runs (48)	At least once* (16)	All runs (48)	At least once* (16)
Model Solution Optimality	**45.83% (22)**	**56.25% (9)**	**47.92% (23)**	**56.25% (9)**
Mathematical model correctness	**37.50% (18)**	**43.75% (7)**	**45.83% (22)**	**56.25% (9)**
Parameter correctness	83.33% (40)	93.75% (15)	85.42% (41)	87.50% (14)
Variable correctness	81.25% (39)	93.75% (15)	95.83% (46)	100.00% (16)
Objective function correctness	54.17% (26)	68.75% (11)	70.83% (34)	87.50% (14)
Constraint correctness	50.0% (24)	62.50% (10)	58.33% (28)	75.00% (12)

Figure 8: Experiment outcomes of the metrics assessing the generated model. (*For the "At least once" column a success was counted if the model generated the correct outcome in at least one out of the three runs.)

The performance of both pipelines was unexpectedly similar. Pipeline 1 resulted in models that reach the optimal solution to the described problem in 45.8% of the runs (22 times) while Pipeline 2 had a slightly higher rate of 47.9% (23 times). This means that there was no statistical evidence found to indicate that Pipeline 2 performed any differently than Pipeline 1, with a p-value of 0.84. Both pipelines solved the same number of problems at least once across the three runs (9), but it is noteworthy that they were not the same problems. As seen in Figure 7, Pipeline 1 reached a model giving a correct solution for the IP2 problem once, while Pipeline 2 did not. Conversely, Pipeline 2 did so for the MIP3 problem, which Pipeline 1 could not achieve. This highlights that while overall success rates are comparable, the specific outcomes were dependent on the solution consistency, which was measured by the MM3 metric. For Pipeline 1 it was 52.1% and for Pipeline 2 it was 56.3%. This suggests that around two semantically different models were generated per problem across the three runs. While this inconsistency might be interpreted as a shortcoming, it implies that the LLM sometimes fails to achieve the correct solution due to randomness. Trying to run the pipeline multiple times and combining the outcomes in an ensemble could potentially mitigate the inconsistency and lead to better results. The scale of the potential improvement can be seen in the "At least once" column in Figure 8. It is also noteworthy that in 5 cases the models generated by the LLM reached optimal solutions despite not being fully correct. The Mathematical Model Correctness metric was successful in 37.5% of runs in Pipeline 1 and 45.8% in Pipeline 2, which is lower than the Model Solution Optimality scores. The mistakes were division by a variable, which could be equal to 0, and errors in inactive elements of objective functions. Those cases are undesirable because it is possible that such a model would not reach the optimal solution when given different input data. Overall, the LLM made more mistakes in generating objective functions and constraints (correct in 50.0% to 70.8% of the runs), than in parameters and variables (correct in 81.3% to 95.8% of the runs).

	Pipeline 1 percentage (count)		Pipeline 2 percentage (count)	
Metric	All runs (48)	At least once* (16)	All runs (48)	At least once* (16)
Equivalent Code Representation of Mathematical Model	79.17% (38)	93.75% (15)	93.75% (45)	100.00% (16)
Executability of Model-Related Code	56.25% (27)	68.75% (11)	79.17% (38)	100.00% (16)
Code Solution Optimality	41.67% (20)	56.25% (9)	41.67% (20)	50.00% (8)

Figure 9: Experiment outcomes of the metrics assessing the generated Pyomo code. (*For the "At least once" column a success was counted if the model generated the correct outcome in at least one out of the three runs.)

Regarding the outcomes of the coding part of the experiment, the semantics of the mathematical model were reflected in the code 79.1% of the time in Pipeline 1 and 93.75% of the time in Pipeline 2. The difference between the two results comes from Pipeline 2 not generating as many indicator functions in the model formulations, which Gemini 1.5 Pro consistently failed to code. Despite representing the semantics, the code was executable 56.3% of the time in Pipeline 1 and 79.2% of the time in Pipeline 2. This discrepancy in the performance is interesting since theoretically the step of generating code was the same in both pipelines. One possible explanation of the better executability of the second pipeline could be that the LLM responses generating the model were shorter and contained less text, however, there could also be other contributing factors, including random odds. Nonetheless, executable code does not necessarily result in the optimal solution, hence the Code Solution Optimality was 41.7% in both pipelines.

Pipeline	Model Solution Optimality % (Count out of 12)			
	IP	LP	MIP	NLP
1	58.3% (7)	66.7% (8)	25.0% (3)	33.3% (4)
2	50.0% (6)	75.0% (9)	33.3% (4)	33.3% (4)

Figure 10: Model solution optimality outcomes grouped by the optimization class.

The results were also grouped by the optimization class, the results of which can be seen in Figure 10. Across both pipelines, the highest success ratio was in the problems of the LP class, followed by the IP, NLP, and MIP, which corresponds to the presumed difficulty of the classes. However, the number of problems of each class included does not allow to draw significant conclusions.

Throughout the experiment, some failure patterns were observed and were grouped by where in the pipeline they occurred. Interestingly, the LLM was failing in mostly the same ways across both pipelines. Firstly, when outlining the parameters, Gemini 1.5 Pro made unnecessary assumptions that were not specified in the problem description. Examples of this include combining parameter values, such as machining capacity and manpower capacity into “production capacity”, which was never mentioned in the description. It also split other parameters into more, such as creating a separate storage capacity for every item instead of having one common storage. Those issues could be mitigated by more precise problem descriptions. To achieve them in a user-friendly way, it could be worth investigating if an interactive approach could work, where the LLM asks to clarify some parts of the description. Another failure mode was the LLM confusing parameters with similar names. A way of preventing this could be the user confirming the available parameters before the LLM proceeds to generate the rest of the model. The failures in defining variables were usually a consequence of wrong parameter definitions. Otherwise, the LLM skipped variables for some components of cost calculations.

Moving on to the failure modes related to the objective function, several patterns not caused by mistakes in the earlier steps have emerged. In Pipeline 1, when faced with a harder problem, the LLM incorrectly tried using indicator, max, and min functions. Curiously, this was not an issue in Pipeline 2. Gemini 1.5 Pro also unnecessarily used absolute values, turning problems non-linear and increasing their difficulty. One possible way to reduce the impact of those patterns could be to optimize the prompt to encourage more direct formulations where possible. In many runs the LLM also struggled to notice nuance in sentences, which it interpreted differently than intended. An example of such behavior could be the model considering the covariance of returns across investments rather than the variance of the returns of each single investment. Another could be applying a discount for products bought over a threshold, rather than applying a discount to all bought products after a threshold is reached. Such small linguistic differences may be difficult to understand by the LLM, which likely was not trained on many relevant examples. This failure mode further emphasizes the need for very precise problem descriptions. The last persistent challenge in the objective function formulation was the tendency of Gemini 1.5 Pro to omit parts of it. For example, in a network question with the objective to maximize throughput to a given node, it would consider just one out of two edges leading to the node. This problem had the data specified in its description, so a possible solution could be to provide it as a dataset. In this way, there would be less information in the description that could confuse the LLM.

In terms of constraint formulation, Gemini 1.5 Pro mainly struggled with robust programming and uncertainty. When asked to allow only a certain probability of exceeding a budget, it would use an undefined probability function P, even though all the data required for the calculations was provided. It also failed when being asked to use error margins by either applying them to the wrong side of the inequality or to the wrong parameters. Those issues may potentially be caused by a small number of examples in the training data. A solution worth investigating could be providing the LLM with more examples of robust programming by fine-tuning. Another difficulty in generating the constraints were the problems with variables including indices representing time steps. In such, Gemini 1.5 Pro would sometimes ignore changes to the time index. This issue may be hard to address without increasing the reasoning capabilities of the LLM itself, however, a self-checking step in the pipeline with the focus on time steps could be investigated.

Lastly, the failures in translating the mathematical formulation of the problem to Python code followed distinct patterns. Implementing indicator, max, and min functions in Pyomo requires defining extra variables. The LLM was not using this approach and would keep the function in the mathematical expression, resulting in Type Errors. When working with indexed variables, Gemini 1.5 Pro would often make mistakes in how it looped over the defined indices, causing Key Errors. The syntax of the used

optimization library also requires two-sided inequalities (e.g. $1 < X < Y$) to be split into two one-sided inequalities ($1 < X$ and $X < Y$), which the LLM never did. Notably, the solvers chosen by Gemini 1.5 Pro were not always appropriate for the optimization class of the problem, however, this was not considered an error in the metrics, as a universal solver could be used and the LLM proposed alternatives. All the issues related to the executability of the generated Python code could be addressed by implementing a self-correcting step, which would be given the previous code, as well as the error message and iteratively alter the code until it is executable or a limit of iterations is reached.

6 Limitations and Future Research

While the results of this project provide significant insights into the performance of Gemini 1.5 Pro on optimization modelling, there were several limitations of the study design that need to be acknowledged. The first one was the small number of problems tested, as compared to other studies. Although the experiment included problems of four different classes, there were few examples of each, making the comparison more qualitative and not statistically significant. This design was intentional, as one of the research goals was to analyze the failure patterns of the LLM in detail, which is a labor-intensive task since the mistakes in model formulations need to be well understood. To achieve more significant results more problems could be gathered and the evaluation could be automated, however, this approach would focus solely on whether the optimal solution was generated, rather than a wider variety of metrics. Another limitation was Gemini API not allowing setting seeds for text generation, which reduced the reproducibility of the results. The inherent randomness of the experiment was partly addressed by running each problem three times, however, as was demonstrated by the Solution Consistency metric reaching 51.2% in pipeline 1, the research could benefit from more problem runs. Increasing the number would also improve the interpretability of the chosen consistency metric, as it performs best with a higher count of solution pairs. A limitation of the coding part of the pipelines was the exclusive use of Pyomo as the optimization library, which was done due to our familiarity with it and to streamline the evaluation process. Allowing the LLM to choose what tools it uses to code the model could bring other results and is an interesting future research direction. Additionally, in some problems, the data was loaded into the generated code manually and its format was adjusted to fit the generated code, which could not be done by business users. Investigating how to effectively make the LLM understand the structure of data would be an interesting research path, but it was beyond the scope of this study. Lastly, the prompts used in the experiment were intentionally made simple and were not optimized for best performance. To address that, a new pipeline using the OPRO technique described by Yang et al. (2023) could be designed, which uses an LLM to try different prompts and find the best one.

Building upon the results of this study, several future research directions could be explored to further enhance the understanding of the performance of LLMs on optimization modelling. One possibility is to not only consider if the solution is optimal, but to check its feasibility and how far is it from the optimum. Sometimes small model mistakes can just slightly affect the objective value, which has not been explored in relevant research. Many use cases could benefit from decision improvements, even if the solutions are not optimal. The experiment also showed that Gemini 1.5 Pro struggled with interpreting some more nuanced elements of problem descriptions, hence it would be interesting to investigate a more interactive approach to an optimization co-pilot, which could ask the user to clarify the more complicated sentences. Such an approach would be harder to robustly experiment on but could address many failure patterns. It would also reflect how interactions between business and optimization experts traditionally work. Moreover, future pipelines could be enhanced by introducing self-checking steps, which would debug the model and code based on the produced error messages. Lastly, the experiment was designed with the assumption that the optimization co-pilot would be used by a business expert with no experience in optimization modelling. However, it would also be valuable to explore how LLMs can assist optimization experts in working more efficiently. In such a scenario, a higher margin of error from the LLM could be acceptable, while still leading to financial and societal benefits.

7 Conclusions

The analysis of the performance of Gemini 1.5 Pro on optimization modelling tasks reveals that the LLM demonstrates a notable capability in formulating mathematical models from natural language descriptions, albeit with limitations. Both experiment pipelines showed comparable performance, achieving model solution optimality in around 45-48% of the runs. The consistency of the answers was moderate, highlighting the influence of randomness. The capabilities varied among optimization classes, with linear programming problems having the highest success rate, though the sample size limits the significance of these results. Translating mathematical models into code also showed variability, with Pipeline 2 (using chain-of-thought) outperforming Pipeline 1 (using a one-step approach) in terms of semantic equivalence and executability, but not in achieving optimal solutions. Failure patterns were consistent across both pipelines, indicating areas where Gemini 1.5 Pro systematically struggles, such as robust programming. Potential improvements, like interactive problem clarification and iterative self-correction, were identified as strategies to enhance the performance. The overall optimality outcomes align with the expectations set by existing literature given the complexity of the problems selected for the dataset. An interesting deviation from the findings of Li et al. (2023) and AhmadiTeshnizi et al. (2023) is the lack of significant differences between tested pipelines. It may suggest that improvements in the Gemini 1.5 Pro as a contemporary LLM compared to older tested models, such as GPT-3.5 and PaLM2, have increased the model's understanding

of issues addressed by pipeline enhancements in those papers. Consequently, basic pipeline improvements, such as chain-of-thought reasoning based on model components, may address simpler issues that the LLM no longer struggles with. Overall, while Gemini 1.5 Pro is capable of designing and solving some optimization models based on natural language problem descriptions, its practical application should be approached with an understanding of its current limitations and areas where human intervention is still crucial. Considering that, there could already be some financial and societal benefits from implementing LLMs into decision-making workflows, especially when working with simpler problems.

Statement on work and originality

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

References

AhmadiTeshnizi, A., Gao, W., & Udell, M. (2023, October 30). *OptiMUS: Optimization Modeling Using MIP Solvers and large language models*. ArXiv.org. https://doi.org/10.48550/arXiv.2310.06116

Ahmed, T., & Choudhury, S. (2024, March 2). *LM4OPT: Unveiling the Potential of Large Language Models in Formulating Mathematical Optimization Problems*. ArXiv.org. https://doi.org/10.48550/arXiv.2403.01342

Amarasinghe, P. T., Nguyen, S., Sun, Y., & Alahakoon, D. (2023, September 29). *AI-Copilot for Business Optimisation: A Framework and A Case Study in Production Scheduling*. ArXiv.org. https://doi.org/10.48550/arXiv.2309.13218

Anil, R., Dai, A. M., Firat, O., Johnson, M., Lepikhin, D., Passos, A., Shakeri, S., Taropa, E., Bailey, P., Chen, Z., Chu, E., Clark, J. H., Shafey, L. E., Huang, Y., Meier-Hellstern, K., Mishra, G., Moreira, E., Omernick, M., Robinson, K., & Ruder, S. (2023, May 17). *PaLM 2 Technical Report*. ArXiv.org. https://doi.org/10.48550/arXiv.2305.10403

Anthropic. (2024, March 4). *Introducing the next generation of Claude*. Www.anthropic.com. https://www.anthropic.com/news/claude-3-family

Benalcazar, P., Malec, M., Kaszyński, P., Kamiński, J., & Saługa, P. W. (2024). Electricity Cost Savings in Energy-Intensive Companies: Optimization Framework and Case Study. *Energies*, *17*(6), 1307. https://doi.org/10.3390/en17061307

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, April 5). *Demystifying Chains, Trees, and Graphs of Thoughts*. ArXiv.org. https://doi.org/10.48550/arXiv.2401.14295

Birge, J., & 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. (1968). *Selected applications of nonlinear programming*. https://apps.dtic.mil/sti/pdfs/AD0679037.pdf

Castillo, E., Conejo, A., Pedregal, P., Garcia, R., & Alguacil, N. (2011, January 1). *Building and Solving Mathematical Programming Models in Engineering and Science*. John Wiley & Sons. https://doi.org/10.1007/978-3-030-97626-2

Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., Barham, P., Chung, H. W., Sutton, C., Gehrmann, S., Schuh, P., Shi, K., Tsvyashchenko, S., Maynez, J., Rao, A., Barnes, P., Tay, Y., Shazeer, N., Prabhakaran, V., & Reif, E. (2022). PaLM: Scaling Language Modeling with Pathways. *ArXiv:2204.02311* . https://arxiv.org/abs/2204.02311

Cornuéjols, G., Peña, J., & Tütüncü, R. (2006). *Optimization Methods in Finance (Vol. 5)*. Cambridge University Press. https://doi.org/10.1017/9781107297340

den Hertog, D., Ergun, O., Goncalves, R., Peters, K., Fleuren, H., Freeman, M., Kavelj, M., & Silva, S. (2019, May 7). *The Nutritious Supply Chain: Optimizing Humanitarian Food Aid*. Optimization-Online.org. https://optimization-online.org/2019/05/7198/

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018, October 11). *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. ArXiv.org. https://arxiv.org/abs/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 (Cornell University)*. https://doi.org/10.48550/arxiv.2401.03244

Gangwar, N., & Kani, N. (2022, December 26). *Highlighting Named Entities in Input for Auto-Formulation of Optimization Problems*. ArXiv.org. https://arxiv.org/abs/2212.13201

Google. (n.d.). *Documentation of Gemini models*. Google AI for Developers. Retrieved May 31, 2024, from https://ai.google.dev/gemini-api/docs/models/gemini

Google. (2024). *Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context*. arxiv.org. https://arxiv.org/pdf/2403.05530

Google, Anil, R., Borgeaud, S., Wu, Y., Alayrac, J.-B., Yu, J., Soricut, R., Schalkwyk, J., Dai, A. M., Hauth, A., Millican, K., Silver, D., Petrov, S., Johnson, M., Antonoglou, I., Schrittwieser, J., Glaese, A., Chen, J., Pitler, E., & Lillicrap, T. (2023, December 18). *Gemini: A Family of Highly Capable Multimodal Models*. ArXiv.org. https://doi.org/10.48550/arXiv.2312.11805

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

Hoffmann, J., Borgeaud, S., Mensch, A., Buchatskaya, E., Cai, T., Rutherford, E., Casas, D. de L., Hendricks, L. A., Welbl, J., Clark, A., Hennigan, T., Noland, E., Millican, K., Driessche, G. van den, Damoc, B., Guy, A., Osindero, S., Simonyan, K., Elsen, E., & Rae, J. W. (2022). Training Compute-Optimal Large Language Models. *ArXiv:2203.15556* . https://arxiv.org/abs/2203.15556

Hsiao, S. (2024, May 14). *Get more done with Gemini: Try 1.5 Pro and more intelligent features*. Google. https://blog.google/products/gemini/google-gemini-update-may-2024/#context-window

Kim, J., & Reed, J. L. (2010). OptORF: Optimal metabolic and regulatory perturbations for metabolic engineering of microbial strains. *BMC Systems Biology*, *4*(1). https://doi.org/10.1186/1752-0509-4-53

Le, H., Wang, Y., Akhilesh Deepak, G., Savarese, S., & Steven. (2022). CodeRL: Mastering Code Generation through Pretrained Models and Deep Reinforcement Learning. *ArXiv*. https://doi.org/10.48550/arxiv.2207.01780

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:1910.13461*. https://arxiv.org/abs/1910.13461

Li, Q., Zhang, L., & Mak-Hau, V. (2023, November 1). *Synthesizing mixed-integer linear programming models from natural language descriptions*. NASA ADS. https://doi.org/10.48550/arXiv.2311.15271

Mistral AI. (2024, April 17). *Cheaper, Better, Faster, Stronger. Continuing to push the frontier of AI and making it accessible to all.* Mistral.ai. https://mistral.ai/news/mixtral-8x22b/

MOSEK. (n.d.). *11.3 Robust linear Optimization — MOSEK Optimization Toolbox for MATLAB 10.2.0*. Docs.mosek.com. Retrieved June 2, 2024, from https://docs.mosek.com/latest/toolbox/case-studies-robust-lo.html

Naveed, H., Khan, A. U., Qiu, S., Saqib, M., Anwar, S., Usman, M., Akhtar, N., Barnes, N., & Mian, A. (2023, August 18). *A Comprehensive Overview of Large Language Models*. ArXiv.org. https://doi.org/10.48550/arXiv.2307.06435

OpenAI. (2022, November 30). *Introducing ChatGPT*. Openai.com. https://openai.com/index/chatgpt/

OpenAI. (2023a, March 14). *GPT-4*. Openai.com. https://openai.com/index/gpt-4-research/

OpenAI. (2023b, November 6). *Introducing GPTs*. Openai.com. https://openai.com/index/introducing-gpts/

Pedroso, J. P., Rais, A., Kubo, M., & Muramatsu, M. (n.d.). *Facility location problems — Mathematical Optimization: Solving Problems using Gurobi and Python*. Scipbook. Retrieved June 2, 2024, from https://scipbook.readthedocs.io/en/latest/flp.html

Poler, R., Mula, J., & Díaz-Madroñero, M. (2014). Operations Research Problems Statements and Solutions. In *Springer eBooks*. https://doi.org/10.1007/978-1-4471-5577-5

Raj, H., Rosati, D., & Majumdar, S. (2023). *Measuring Reliability of Large Language Models through Semantic Consistency*. https://arxiv.org/pdf/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, March 26). *NL4Opt Competition: Formulating Optimization Problems Based on Their Natural Language Descriptions*. ArXiv.org. https://doi.org/10.48550/arXiv.2303.08233

Sarker, L., Downing, M., Desai, A., & Bultan, T. (2024, April 1). *Syntactic Robustness for LLM-based Code Generation.* ArXiv.org. https://doi.org/10.48550/arXiv.2404.01535

Schulz, J. (2021). Applying Mathematical Optimization in Practice. *Operations Research Forum*, *2*(1). https://doi.org/10.1007/s43069-020-00046-9

Shazeer, N., Mirhoseini, A., Maziarz, K., Davis, A., Le, Q., Hinton, G., & Dean, J. (2017). Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer. *ArXiv:1701.06538* . https://arxiv.org/abs/1701.06538

StemEZ. (n.d.). *Problem 04 – 0177*. Stemez.com. Retrieved June 2, 2024, from https://stemez.com/subjects/science/1HOperationsReseach/1HOperationsReseach/1HOperationsResearch/1H04-0177.htm

Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., Bikel, D., Blecher, L., Ferrer, C. C., Chen, M., Cucurull, G., Esiobu, D., Fernandes, J., Fu, J., Fu, W., & Fuller, B. (2023, July 19). *Llama 2: Open Foundation and Fine-Tuned Chat Models*. ArXiv.org. https://doi.org/10.48550/arXiv.2307.09288

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017, June 12). *Attention Is All You Need*. ArXiv.org. https://arxiv.org/abs/1706.03762

Wasserkrug, S., Boussioux, L., den Hertog, D., Mirzazadeh, F., Birbil, I., Kurtz, J., & Maragno, D. (2024, February 26). *From Large Language Models and Optimization to Decision Optimization CoPilot: A Research Manifesto*. Arxiv.org. https://arxiv.org/html/2402.16269v1#bib.bib15

Wright, S. J. (2024). Optimization. In *Encyclopedia Britannica*. https://www.britannica.com/science/optimization

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.net. https://openreview.net/forum?id=HobyL1B9CZ

Yang, C., Wang, X., Lu, Y., Liu, H., Le, Q. V., Zhou, D., & Chen, X. (2023, September 6). *Large Language Models as Optimizers*. ArXiv.org. https://doi.org/10.48550/arXiv.2309.03409

Zak, E. J. (2024). How to Solve Real-world Optimization Problems. In *SpringerBriefs in operations research*. Springer International Publishing. https://doi.org/10.1007/978-3-031-49838-1

Zhao, W. X., Zhou, K., Li, J., Tang, T., Wang, X., Hou, Y., Min, Y., Zhang, B., Zhang, J., Dong, Z., Du, Y., Yang, C., Chen, Y., Chen, Z., Jiang, J., Ren, R., Li, Y., Tang, X., Liu, Z., & Liu, P. (2023). A Survey of Large Language Models. *ArXiv:2303.18223 [Cs]*. https://arxiv.org/abs/2303.18223

Appendix

Appendix A: Project results overview and GitHub repository

Metric	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
Model Solution Optimality	47.92% (23)	45.83% (22)	16.67% (8)	20.83% (10)	**54.17% (26)**	47.92% (23)	22.92% (11)	20.83% (10)
Mathematical model correctness	41.67% (20)	37.5% (18)	2.08% (1)	6.25% (3)	**45.83% (22)**	**45.83% (22)**	14.58% (7)	8.33% (4)
Parameter correctness	**89.58% (43)**	83.33% (40)	85.42% (41)	83.33% (40)	81.25% (39)	85.42% (41)	72.92% (35)	85.42% (41)
Variable correctness	83.33% (40)	81.25% (39)	37.5% (18)	50.0% (24)	89.58% (43)	**95.83% (46)**	54.17% (26)	54.17% (26)
Objective function correctness	**85.42% (41)**	54.17% (26)	52.08% (25)	50.0% (24)	75.0% (36)	70.83% (34)	58.33% (28)	47.92% (23)
Constraint correctness	43.75% (21)	50.0% (24)	8.33% (4)	10.42% (5)	50.0% (24)	**58.33% (28)**	22.92% (11)	16.67% (8)
Equivalent Code Representation of Mathematical Model	**93.75% (45)**	79.17% (38)	83.33% (40)	60.42% (29)	85.42% (41)	**93.75% (45)**	83.33% (40.0)	58.33% (28)
Executability of Model-Related Code	72.92% (35)	56.25% (27)	41.67% (20)	56.25% (27)	62.5% (30)	**79.17% (38)**	27.08% (13)	54.17% (26)
Code Solution Optimality	41.67% (20)	41.67% (20)	16.67% (8)	14.58% (7)	**45.83% (22)**	41.67% (20)	20.83% (10)	16.67% (8)

Appendix Figure 1: Comparison of the metrics' results for the models tested in the project (with the highest score for each metric in bold)

The results of the entire project, including different LLMs tested can be found in the shared GitHub repository: https://github.com/RiVuss/LLMsForOptimizationModelling/tree/main

Appendix B: IP2 Problem data

Problem description:

"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 investment strategies and the uncertain budget. You can take 2 as the risk parameter r."

Reference Mathematical formulation:

• x_i: Binary variable where it is 1 if investment i is selected, 0 otherwise, where i = 1, 2, ..., 10

• a_i: Cost of investing into investment i, where i = 1, 2, ..., 10

• μ_i: Expected return on investment i

• σ^2_i: Variance of the returns

• r: Risk parameter (=2)

• z_j: Binary variable where it equals 1 if scenario j is chosen, and 0 otherwise, where j = 1, 2, 3, 4

• ε: The tolerance for risk (=0.4)

$$\max\left(\sum_{i=1}^{10}\mu_i x_i - r\sum_{i=1}^{10}\sigma_i^2 x_i\right)$$

Subject to:

$$\sum_{i=1}^{10}\tilde{a}_i x_i \le 1000 + (1-z_1)M,$$
$$\sum_{i=1}^{10}\tilde{a}_i x_i \le 1100 + (1-z_2)M,$$
$$\sum_{i=1}^{10}\tilde{a}_i x_i \le 900 + (1-z_3)M,$$
$$\sum_{i=1}^{10}\tilde{a}_i x_i \le 1200 + (1-z_4)M,$$
$$0.55z_1 + 0.4z_2 + 0.04z_3 + 0.01z_4 \ge 1-\epsilon,$$
$$z_1, z_2, z_3, z_4 \in (0,1),$$
$$M = \sum_{i=1}^{10} a_i.\text{where M is 2100}$$

Reference Python code:

```
from pyomo.environ import *
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

#Generate Data for Mean-Variance Model
df = pd.read_csv("investments_data.csv",sep=";", header=None)
headers = []
for i in range(len(df.columns)):
    headers.append("Investment"+str((i+1)))
df.columns = headers

plt.plot(df["Investment1"])
plt.plot(df["Investment2"])
plt.plot(df["Investment3"])
plt.plot(df["Investment4"])
plt.plot(df["Investment5"])
plt.plot(df["Investment6"])
plt.plot(df["Investment7"])
plt.plot(df["Investment8"])
plt.plot(df["Investment9"])
plt.plot(df["Investment10"])
```

```
means = df.mean(0)
variances = df.var(0)

investment_costs = {
    1: 80,
    2: 340,
    3: 410,
    4: 50,
    5: 180,
    6: 221,
    7: 15,
    8: 348,
    9: 191,
    10: 225
}
#THESE ARE ALREADY IN THE CODE AS VALUES, BUT MAY BE USEFUL

# create a model

model = ConcreteModel()

# flow variables
model.x1 = Var(domain=Binary)
model.x2 = Var(domain=Binary)
model.x3 = Var(domain=Binary)
model.x4 = Var(domain=Binary)
model.x5 = Var(domain=Binary)
model.x6 = Var(domain=Binary)
model.x7 = Var(domain=Binary)
model.x8 = Var(domain=Binary)
model.x9 = Var(domain=Binary)
model.x10 = Var(domain=Binary)

model.z1 = Var(domain=Binary)
model.z2 = Var(domain=Binary)
model.z3 = Var(domain=Binary)
model.z4 = Var(domain=Binary)

r=2
eps = 0.4

# declare objective
model.profit = Objective(expr = (means[0]-r*variances[0])*model.x1 +
(means[1]-r*variances[1])*model.x2 + (means[2]-r*variances[2])*model.x3 \
                        + (means[3]-r*variances[3])*model.x4 + (means[4]-
r*variances[4])*model.x5 + (means[5]-r*variances[5])*model.x6 \
                            + (means[6]-r*variances[6])*model.x7 + (means[7]-
r*variances[7])*model.x8 + (means[8]-r*variances[8])*model.x9 \
                                 + (means[9]-r*variances[9])*model.x10 ,
sense=maximize)
```

```
# declare constraints
model.constrA = Constraint(expr = 80*model.x1 + 340*model.x2 + 410*model.x3 +
50*model.x4 + 180*model.x5 + 221*model.x6 + \
                           15*model.x7 + 348*model.x8 + 191*model.x9 +
225*model.x10 + 2100*model.z1<= 1000+2100)

model.constrB = Constraint(expr = 80*model.x1 + 340*model.x2 + 410*model.x3 +
50*model.x4 + 180*model.x5 + 221*model.x6 + \
                               15*model.x7 + 348*model.x8 + 191*model.x9 +
225*model.x10 + 2100*model.z2<= 1100+2100)

model.constrC = Constraint(expr = 80*model.x1 + 340*model.x2 + 410*model.x3 +
50*model.x4 + 180*model.x5 + 221*model.x6 + \
                           15*model.x7 + 348*model.x8 + 191*model.x9 +
225*model.x10 + 2100*model.z3<= 900+2100)

model.constrD = Constraint(expr = 80*model.x1 + 340*model.x2 + 410*model.x3 +
50*model.x4 + 180*model.x5 + 221*model.x6 + \
                               15*model.x7 + 348*model.x8 + 191*model.x9 +
225*model.x10 + 2100*model.z4<= 1200+2100)

model.constrE = Constraint(expr = 0.55*model.z1 + 0.4*model.z2 + 0.04*model.z3
+ 0.01*model.z4 >=1-eps)

# solve
print("\\nSolution")
results = SolverFactory('glpk').solve(model)
#results.write()
if results.solver.status== 'ok':
    model.pprint()

# # display solution
print('\nProfit = ', model.profit())
```

Reference optimal solution:

Investments selected (xi) = x3, x6, x8

Scenarios selected (zj) = z1, z2

Profit: 225.96748959221745

Appendix C: Characteristics of the problems in the dataset

Problem	Domain	Characteristics
LP1	Production planning	Capacity constraints, ratio constraints
LP2	Production planning	Capacity constraints, parameters in a dataset, robust programming
LP3	Production planning	Capacity constraints, cost calculations
LP4	Production planning	Capacity constraints, variable cut-off
IP1	Assignment problem (medical)	Capacity constraints, equality, parameters in a dataset
IP2	Portfolio optimization	Big-M capacity constraints, chance constraint, mean-variance optimization, parameters in a dataset
IP3	Assignment problem (bin packing)	Capacity constraint, logic constraints, equality, parameters in a dataset
IP4	Workforce scheduling + resource allocation	Capacity constraints, priority factors, target deviation, equalities, logic constraints
MIP1	Facility location	Capacity constraint, logic constraint, parameters in a dataset
MIP2	Production planning	Capacity constraints, time steps, parameters in a dataset
MIP3	Production planning	Capacity constraints, logic constraint, time steps, robust programming, parameters in a dataset
MIP4	Network flow	Capacity constraints, time-steps, equalities, data generating function, logic constraint, parameters in a dataset
NLP1	Production planning	Capacity constraint, ratio constraint, different units, unnecessary data
NLP2	Production planning	Capacity constraint
NLP3	Supplier selection	Capacity constraints, logic constraints, equality, piece-wise function
NLP4	Process optimization	Capacity constraints, equalities, robust programming, parameters in a dataset