Large Language Models as Optimizers for Business Applications

A Performance Analysis of Mixtral 8x22B

By Nathan Ferchtandiker (13986163)

University of Amsterdam
BSc Business Analytics

Supervised by Donato Maragno

27.06.2024

Abstract

Optimization is a powerful collection of mathematical techniques able to solve complex problems that possess a set of optimal solutions bound to a set of constraints. However, current utilization of optimization in business applications is limited by the availability of experts capable of solving business-level optimization problems. Contrarily, recent achievements in research related to Large Language Models (LLMs) pose an opportunity to abstract away complex optimization knowledge and thus offer the potential to contribute towards a higher utilization of optimization in business applications. Previous attempts at solving optimization problems with LLMs have yielded promising results but were mostly limited to simple optimization problems that are not representative of business applications. This research investigates the performance of Mixtral 8x22B, an LLM published in April 2024 by Mistral AI, when provided with business-level optimization problem descriptions in natural language and asked to generate associated mathematical formulations and code. To achieve this, a dataset of 16 business-level optimization problems was created. In addition, two pipelines were developed that serve the purpose of guiding Mixtral through the solving process. Finally, to quantify the performance, several metrics were defined and measured. Results suggest that Mixtral struggles with the identification of unwritten and implied model components that are apparent to human experts. In addition, Mixtral failed to identify binary indicator variables in almost all cases and was thus not able to incorporate complex logic in its generated models. Furthermore, its responses were semantically inconsistent across several trials. Thus, the combination of simple pipelines and Mixtral 8x22B proved to be unreliable as a fully autonomous optimizer agent that is capable of autonomously solving business-level optimization problems.

Table of Content

1. Introduction

Optimization can be defined as a set of mathematical methods to solve quantitative problems with an optimal solution that adheres to a set of constraints (Stephen, 2024). It has been used in many business areas such as aviation, finance, healthcare manufacturing and more (Cvetkoska, 2017). Examples include the optimization of truck delivery schedules (Stok, 2022) or optimal allocation of Covid testing centers (Abdin et al. 2023). Thus, optimization can be an essential tool for creating and improving business processes and therefore lead to decreased costs, increased profits, and general business growth (Cvetkoska, 2017). However, several issues that limit the usage of optimization in a business context today. First, the lack of knowledge of decision-makers about the existence of optimization modeling and its benefits is a limiting factor (Cvetkoska, 2017). If managers do not know optimization modeling, they can not utilize it in their operations. Therefore, decision-makers should be educated about the existence of this framework to further propagate its benefits. Second, businesses currently utilize optimization in five different steps, namely defining the problem, formulating it as a mathematical model, replicating the model in a modeling software, tuning the model for efficiency and validating results (Wasserkrug et al., 2024). Naturally, these steps require a high level of domain knowledge in the business and modelling context. Therefore, complex business-level optimization problems are usually solved by experts in an expensive and time-consuming process (Wasserkrug et al., 2024). Other methods that led to faster but less optimal results are often preferred (Cvetkoska, 2017).

Contrarily, Large Language models (LLMs) can perform a multitude of text-processing tasks when provided with natural language as input. As an example, LLMs were previously used to answer general knowledge questions (Devin et al., 2018) or text translation (Brown et al., 2020). As such, LLMs offer the opportunity to support business stakeholders in the decision-making process when utilized as optimization solvers and thus leverage its benefits for business due to the possibility of abstracting away any expert knowledge (Fan et al., 2024; Wasserkrug et al., 2024). Thus, by leveraging LLMs, optimization can be made more accessible for solving business problems leading to the positive outcomes mentioned above (Wasserkrug et al., 2024). Many investigations have been made on using LLMs to solve general optimization problems of various difficulty and realism (AhmadiTeshnizi et al., 2023; Ahmed & Choudhury, 2024; Fan et al., 2024; Gangwar & Kani, 2023; Li et al., 2023; Ramamonjison, 2023a; Ramamonjison, 2023b; Xiao et al., 2023; Yan et al., 2023). Usually, the solving process with LLMs starts with a problem description in natural language and is divided into multiple subprocesses that lead to a mathematical formulation or even code that calculates a solution (AhmadiTeshnizi et al., 2023). However, most research conducted thus far has focused on leveraging LLMs for solving classic optimization commonly found in textbooks. Two issues arise from this. First, the level of complexity of classic problems is lower because complex processes and mechanisms are simplified, and the scale of the problems is typically a lot smaller. Second, the textbook problems are potentially part of the training set used to determine the parameters of the LLM. Business-related optimization problems on the other hand are often unique to each business and application which implies that these problems can not be part of the training set of the LLM. This again implies that solving optimization problems found in business applications with LLMs is more difficult as it requires the LLM to transfer learned knowledge to an example it has never seen before. Therefore, it is unlikely that the achieved results can seamlessly be applied to business-level problems.

2. Literature Review

2.1 Background on Large Language Models

Language models are neural networks that predict the next token conditioned on previously generated or fed tokens. They are built on an architecture called *transformers* that encode and decode tokens and sequences of tokens into a high-dimensional vector space (Bahdanau et al., 2016). The *attention mechanism* allows language models to give specific sections of previously seen or generated tokens a higher weight when deciding which token to choose next, thus allowing language models to understand the context of a task (Vaswani et al., 2017). Large language models are characterized by their large number of tunable parameters in the underlying neural network, nowadays in the billions or trillions. A large number of parameters was associated with better task-agnostic performance (Brown et al., 2020).

Mixtral 8x22B (Mistral AI, 2024) is an open-source LLM introduced in April 2024. Mixtral 8x22B extends the Mistral 7B model (Jiang et al. 2023) by utilizing the *Mixture of Experts* architecture which implies that the LLM has access to specialized model components that are combined during inference depending on the context given by previous tokens (Jiang et al., 2024). Thanks to this architecture, Mixtral 8x22B is computationally more efficient and achieves a better cost ratio during inference on the MMLU dataset (Hendrycks et al., 2020) than comparable models (Mistral AI, 2024).

LLMs can be adapted in various ways to improve the performance on a specific task. First, the model weights and architecture can be adjusted in a process referred to as *Fine-Tuning* (Minaee et al., 2024). LLMs can be fine-tuned by taking the pre-trained LLM and a new training set of answer-response pairs. Then, the LLM is retrained on the reference dataset and its parts of the weights are updated. Common fine-tuning techniques include but are not limited to *Low Rank Adapters* (Hu et al., 2021), *Transfer Learning* (Chronopoulou et al., 2017) and *Reinforcement Learning with Human Feedback* (Ouyang et al., 2022). Fine-tuning requires a lot less computation power and data than pretraining of language models but generally requires at least a few thousand examples and high computation demands (Microsoft, 2024). Second, the model may remain original to the pre-trained language model, but the input can be adjusted such that the LLM has a deeper understanding of the task which is commonly referred to as *Prompt Engineering*. Besta et al. (2024) presented multiple prompt topologies, including *Chain of Thought*, *Tree of Thought* and *Graph of Thought* and found that when LLMs were prompted to solve tasks by breaking them down into subtasks and solving the subtasks separately, an increase in accuracy was achieved. For optimization, this can mean that breaking the generation of a mathematical model into the subtasks of identification of parameters, variables, objective function, and constraints as described in the methodology may increase performance. *Chain of Thought* is a topology where subtasks are solved separately and in sequential order. *Tree of Thought* extends the *Chain of Thought* topology by allowing branching and parallel processing of subtasks. Finally, the *Graph of Thought* topology extends the *Tree of Thought* topology by allowing branches to be merged and thus combining results from several branches. Generally, the determination of subtasks, branches and solving order can be determined by the user or the LLM itself. *In-context learning* describes a prompt technique that provides the LLM with examples of the task to be solved (Dong et al., 2023). In the case of solving Optimization problems with LLMs, this implies that the LLM is fed with a few examples of problem descriptions, matching mathematical models and codes representing the mathematical models. Additional prompting techniques exist. Finally, additional task-specific techniques exist such as *Retrieval-Augmented Generation* (Lewis et al., 2020) which utilize LLMs as a part of a more complex system to generate a text exist. Due to the inherent complexity of *fine-tuning* and task-specific techniques, this study only utilized *Chain of Thought* prompting as described in the Methodology section.

2.2 Previous Work on Large Language Models as Optimizers

Previous research has made significant progress in the utilization of LLMs to solve optimization problems. NL4Opt is a competition in which research groups competed to extract optimization model information from natural language descriptions of optimization problems to improve the utilization and accessibility of optimization modelling among non-experts (Ramamonjison, 2023b). Given the text description of a problem, the participating teams were asked to utilize LLMs to perform e*ntity recognition* to identify model-relevant features (subtask 1) and create a mathematical formulation (subtask 2). However, this competition has been criticized for the lack of complex business-level problems in their dataset (Xiao et al., 2023). Gangwar & Kani (2023) compose the winning team of the NL4Opt competition. They used *fine-tuning* and *entity recognition* as they first parsed the problem description to construct a tagged representation in which model components were highlighted. They found that applying all model component tags at once is more effective than categorizing them by component type and performing the tagging sequentially which is contrary to approaches such as *Chain of Thought* that argue that breaking a problem down into subproblems increases performance. Ahmed & Choudhury (2024) have shown that *in-context learning* increased the accuracy of mathematical models generated by various LLMs as they found that providing examples of problem description and mathematical model pairs provides the LLM with more context on the task. However, they also showed that this is only true for larger LLMs due to context limitations of smaller models. In addition, it was shown that LLMs can convert mathematical models into code that can solve the represented model (Ramamonjison et al. 2023a). They developed *LaTexSolver*, a software for translating mathematical formulations written in LaTex into solver code by leveraging LLMs. They argue that by utilizing LLMs in this way, a leaner workflow can be achieved as optimization problems do not need to be stored and adjusted in their mathematical and code representation. Their software leverages *entity recognition* and *fine-tuning* to identify model components accurately. Unfortunately, Ramamonjison et al. (2023a) did not publish any evaluations of their method.

In addition to participants in the NL4Opt challenge, research teams have derived additional methodologies and results. Contrary to the results of Gangwar & Kani (2023), Xiao et al. (2023) found that *Chain of Thought* prompting increased the accuracy of generated mathematical models. They created several agents that specialized in subtasks. They instructed their agents through *in-context learning*-as each agent had access to a database of task-specific examples. Next, the agents were autonomously chained together to generate a mathematical model and write code. They found that this approach increased the accuracy of the mathematical models between 10% and 20%. Li et al. (2023) also utilized *in-context learning* as they provided LLM solvers with templates of model components and found an increase in the accuracy of the mathematical model.

However, there has also been criticism of LLMs as optimization solvers. More specifically, Yan et al. (2023) argued that because LLMs draw tokens from a probability distribution, there is an inherent chance of error that will ultimately lead to the generation of flawed mathematical models, especially for large and complex problems commonly found in business applications. In addition, it is expected that businessusers may prefer not to follow a strict input structure. However, it has been shown that imprecise and inconsistent problem descriptions may be an unscalable input format for LLMs (AhmadiTeshnizi et al. 2023). Thus, the generation of a consistent and reliable problem description may issue another challenge for autonomous LLM optimizers.

In summary, various approaches have been utilized and tested to improve the performance of various LLMs on textbook and business-level problems. Generally, the variety of input formats of problem descriptions was shown to be problematic as it was shown to influence the performance during solving (AhmadiTeshnizi et al., 2023; Yan et al., 2023). More structured input formats typically require the manual identification of parameters, variables, objective function and constraints which is contrary to an autonomous solving process for business stakeholders. In addition, the complexity and scale of business-level problems ar likely to affect the performance of the LLMs due to a limited context size (AhmadiTeshnizi et al., 2023; Ahmed & Choudhury 2024). However, given the recent explorations of *fine-tuning* and *in-context learning* and resulting achievements, the field continues to move towards more reliable LLM optimizers.

2.3 Contributions

Most of the previous research in this field has been focused on popular LLMs such as GPT or Llama. Little to no research has been conducted on less popular models such as Mixtral 8x22B (Mistral AI, 2024) which will be investigated in this study. In addition, the primary goal of previous research was to optimize the accuracy and performance of LLMs as autonomous optimizers. However, to understand how to optimize the potential pipelines, a prior understanding of failure patterns across various problem classes and domains must be established. This research aims to establish these insights. More specifically, the main objective of this research is to identify failure patterns in the conversion process between problem description in natural language, mathematical formulation and code conducted by Mixtral 8x22B and thus aims to contribute towards an understanding of the capabilities of LLMs to solve business-level optimization problems. To achieve this, a set of 16 business-level optimization problems containing a problem description, mathematical formulation and code were defined. Furthermore, two different pipelines were utilized that served the purpose of guiding the LLM through the solving process. Both pipelines provide Mixtral 8x22B with the problem description in natural language, instructions to convert it into a mathematical model and instructions to convert the mathematical model into code to obtain a solution. In addition, metrics were defined to quantify the performance of the model. Both are described in more detail in the methodology section. Resulting of this is an analysis of the failure patterns, a judgement of the Mixtral 8x22's ability to generate mathematical models and codes to optimization problems and a set of suggested research that tackles the failure patterns identified in this study.

3. Methodology

3.1 Data Description

The dataset used to conduct the experiments was curated to create a set of problems that varies across problem classes and domains to allow for the identification of failure patterns. One of the required characteristics of the chosen problems was that they should reflect real business use cases. Previous research has used datasets such as the NL4Opt dataset (Ramamonjison et al., 2023) to investigate whether LLMs are capable of formulating and solving mathematical optimization models (Ahmed & Choudhury, 2024; Xiao et al., 2024). But as mentioned by Ramamonjison et al. 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 fairly 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 and introduced some logic constraints as well as 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 structured natural language optimization problems in the domain of linear and mixed integer programming. Amarasinghe et al. (2023) investigated the abilities of a fine-tuned LLM to solve 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 Mixtral 8x22B 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 research consists of a more diverse set of mathematical optimization problems. Given the time span and scope of this study, 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 (Hart et al., 2011) code calculating the optimal solution and optional data files containing parameter values. While the natural language description served as input for the LLM, the mathematical and respective Pyomo codes served evaluation purposes only.

Problem	**Domain**	**Characteristics**
IP1	Assignment problem	Capacity constraints, ratio constraints
IP2	Portfolio Optimization	Capacity constraints, parameters in a dataset, robust programming
IP3	Assignment problem	Capacity constraints, cost calculations
IP4	Scheduling	Capacity constraints, variable cut-off
LP1	Production planning	Capacity constraints, equality, parameters in a dataset
LP2	Production planning	Big-M capacity constraints, chance constraint, mean-variance optimization, parameters in a dataset
LP3	Production planning	Capacity constraint, logic constraints, equality, parameters in a dataset
LP4	Production planning	Capacity constraints, priority factors, target deviation, equalities, logic constraints
MIP1	Fascility allocation	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

Table 1: Overview of Problem Classes, Domains and Characteristics.

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 usage 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 remarked that the LLMs used in their study struggled with unit conversions used to introduce more complexity into the dataset used in the NL4OPT competition. Due to the nature of the use cases, some of the problem descriptions were 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.

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.).

The training data of Mixtral 8x22B may include information up to April 2024 (Mistral AI, 2024). 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. Both of these factors help mitigate the risk of using optimization problems as part of this research that the LLM has been trained on. Overall, using a set of such diverse optimization problems aids in more realistically assessing whether Mixtral could be suitable for applications in real business settings, for example as Decision Optimization CoPilot as envisioned by Wasserkrug et al. (2024).

3.2 Experiment Pipelines

Previous literature has experimented with different pipeline designs for the task of generating mathematical optimization problems with LLMs (AhmadiTeshnizi et al., 2023; Ahmed & Choudhury, 2024; Fan et al., 2024; Gangwar & Kani, 2023; Li et al., 2023; Ramamonjison, 2023a; Ramamonjison, 2023b; Xiao et al., 2023; Yan 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 was chosen for its support of a wide range of problem types and the researcher's familiarity with its syntax.

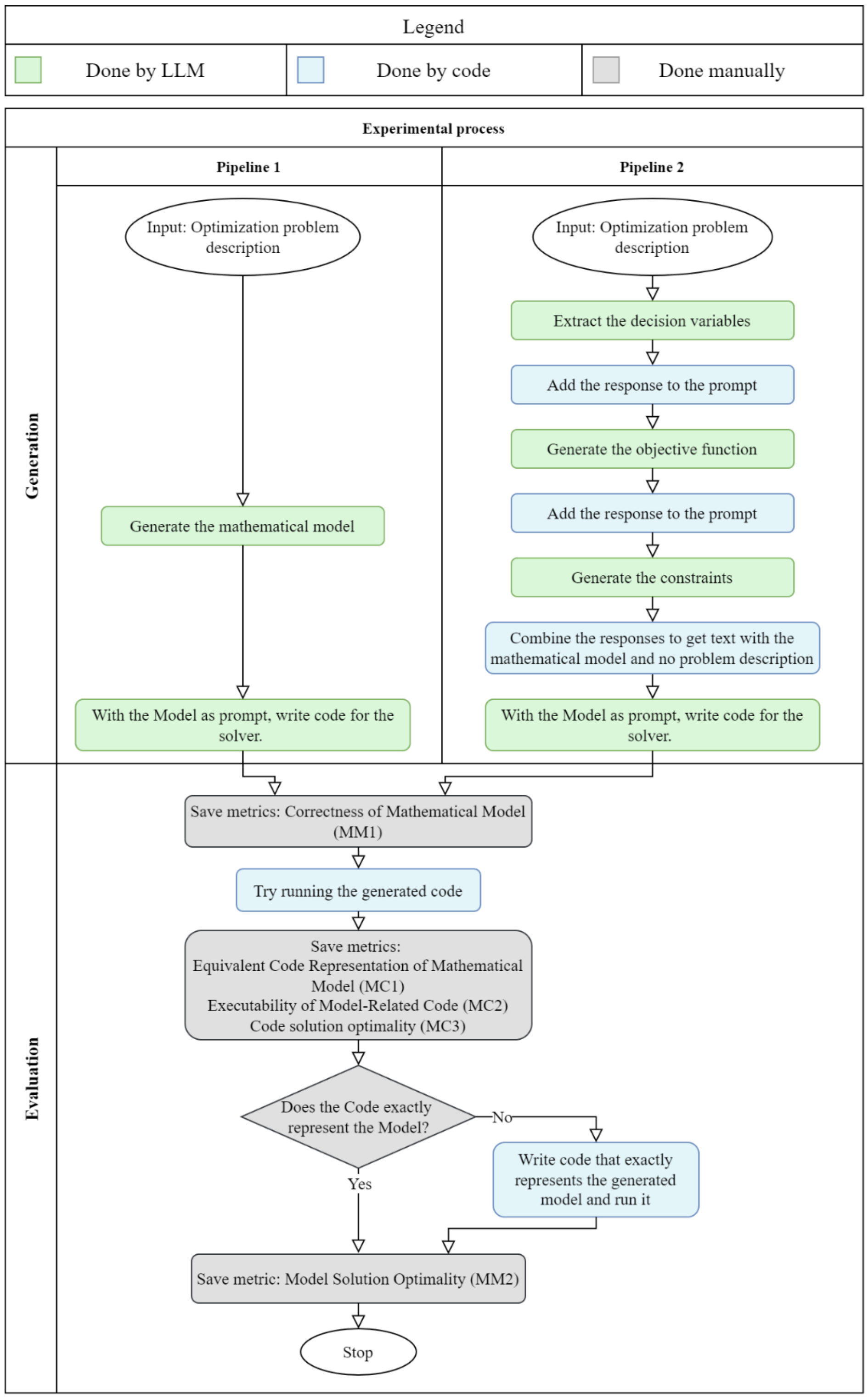

Legend
Done by LLM
Done by code
Done manually
Experimental process
Pipeline 1
Pipeline 2
Generation
Evaluation
Input: Optimization problem description
Generate the mathematical model
With the Model as prompt, write code for the solver.
Input: Optimization problem description
Extract the decision variables
Add the response to the prompt
Generate the objective function
Add the response to the prompt
Generate the constraints
Combine the responses to get text with the mathematical model and no problem description
With the Model as prompt, write code for the solver.
Save metrics: Correctness of Mathematical Model (MM1)
Try running the generated code
Save metrics: Equivalent Code Representation of Mathematical Model (MC1) Executability of Model-Related Code (MC2) Code solution optimality (MC3)
Does the Code exactly represent the Model?
No
Yes
Write code that exactly represents the generated model and run it
Save metric: Model Solution Optimality (MM2)
Stop

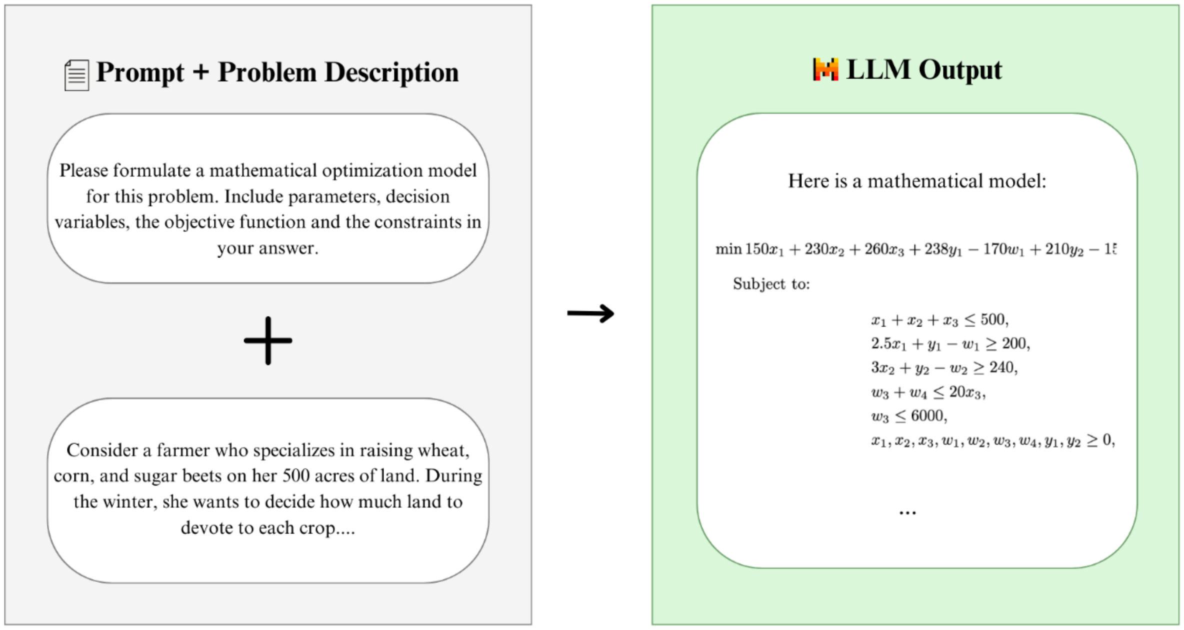
Figure 1: Overview of Pipeline Workflows for Pipeline 1 and Pipeline 2.

3.2.1 Pipeline 1

The idea for the first pipeline was to emulate a simple LLM prompt, 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. 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 running this step can be seen in Figure 2.



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

+

Consider a farmer who specializes in raising wheat, corn, and sugar beets on her 500 acres of land. During the winter, she wants to decide how much land to devote to each crop....

Here is a mathematical model:

$$\min 150x_1 + 230x_2 + 260x_3 + 238y_1 - 170w_1 + 210y_2 - 1$$

Subject to:

$$\begin{aligned} &x_1 + x_2 + x_3 \leq 500, \\ &2.5x_1 + y_1 - w_1 \geq 200, \\ &3x_2 + y_2 - w_2 \geq 240, \\ &w_3 + w_4 \leq 20x_3, \\ &w_3 \leq 6000, \\ &x_1, x_2, x_3, w_1, w_2, w_3, w_4, y_1, y_2 \geq 0, \end{aligned}$$

...

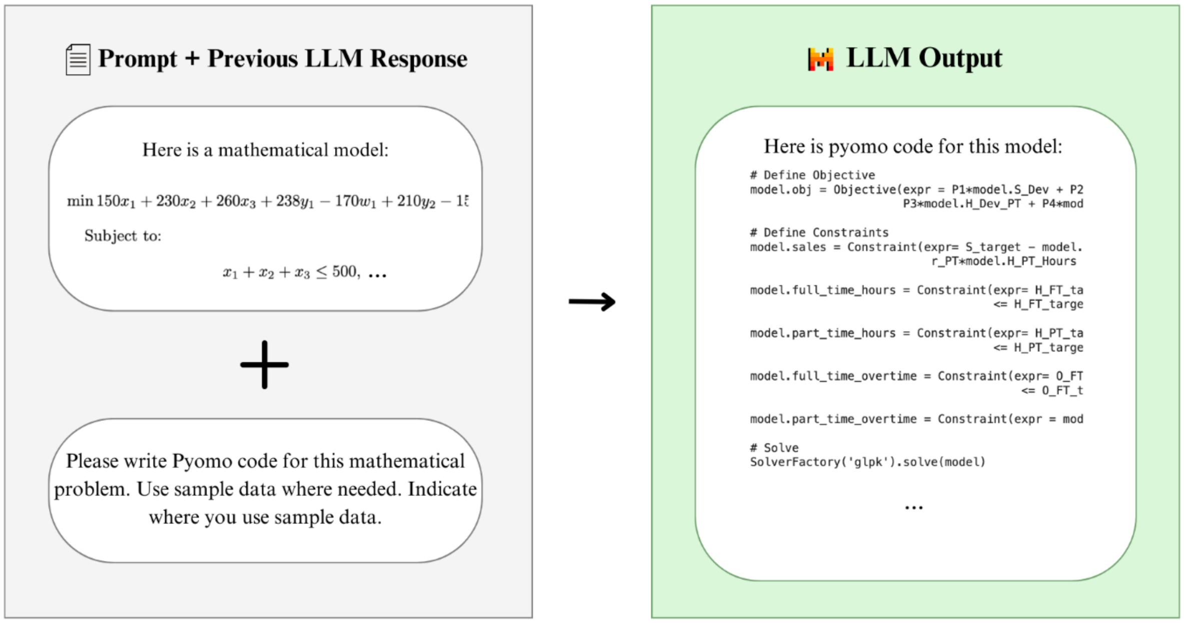
Figure 2: The Modelling Step of the Experiment Pipeline 1.

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 2):

"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 minor mistakes or inactive constraints. The prompt also included 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.



Here is a mathematical model:

$$\min 150x_1 + 230x_2 + 260x_3 + 238y_1 - 170w_1 + 210y_2 - 1$$

Subject to:

$$x_1 + x_2 + x_3 \leq 500, \ldots$$

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

Here is pyomo code for this model:

```
# Define Objective
model.obj = Objective(expr = P1*model.S_Dev + P2
                      P3*model.H_Dev_PT + P4*mod

# Define Constraints
model.sales = Constraint(expr= S_target - model.
                        r_PT*model.H_PT_Hours

model.full_time_hours = Constraint(expr= H_FT_ta
                                   <= H_FT_targe

model.part_time_hours = Constraint(expr= H_PT_ta
                                   <= H_PT_targe

model.full_time_overtime = Constraint(expr= O_FT
                                      <= O_FT_t

model.part_time_overtime = Constraint(expr = mod

# Solve
SolverFactory('glpk').solve(model)
```

...

Figure 3: The Coding Step of the Experiment Pipeline 1

.

3.2.2 Pipeline 2

The second pipeline was designed to be a more advanced use of LLMs. The problem description input remained unaltered to 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 the task down 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. To achieve this, the Mixtral required special instructions which enabled it to focus on specific subtasks which was achieved with the following prompts:

1. *"Please formulate a mathematical optimization model for this problem.*
   It is important that you do this by following these steps:
   1. Define parameters and variables
   2. Define the objective function
   3. Define the constraints

Please note that each answer is always dedicated to one step. Therefore, do not provide extra information than what is asked. The answers will be manually combined by the user to create the full model."

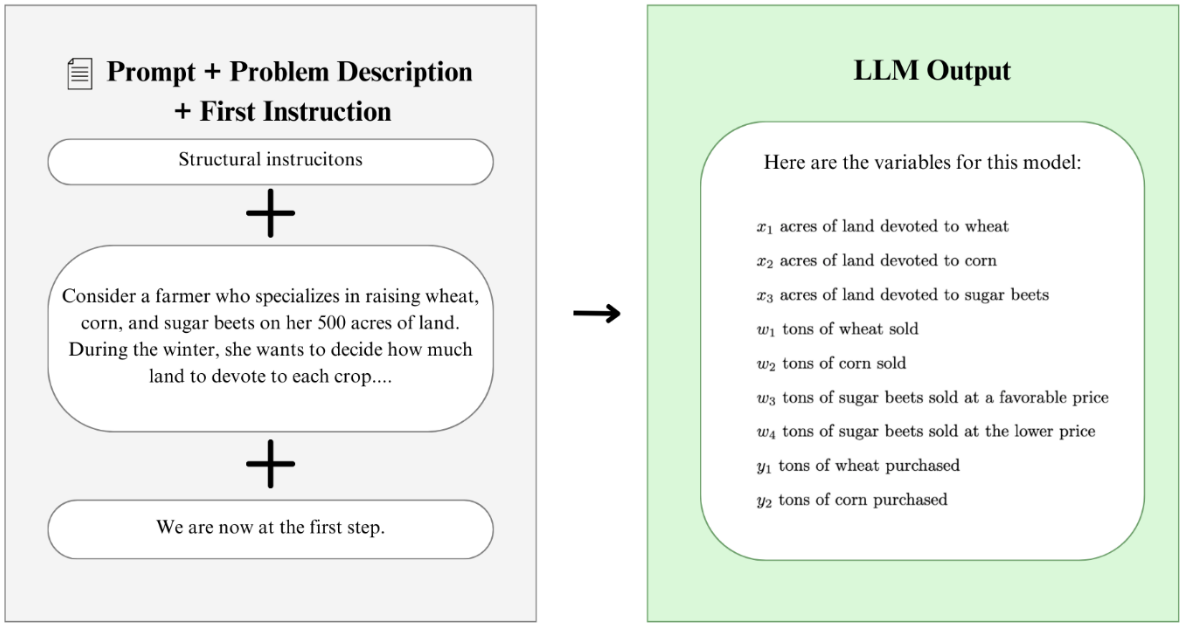
2. Problem Description

3. *" We are now at the first step."*

4. *"We are now at the second step."*
   + LLM's response to prompt 3

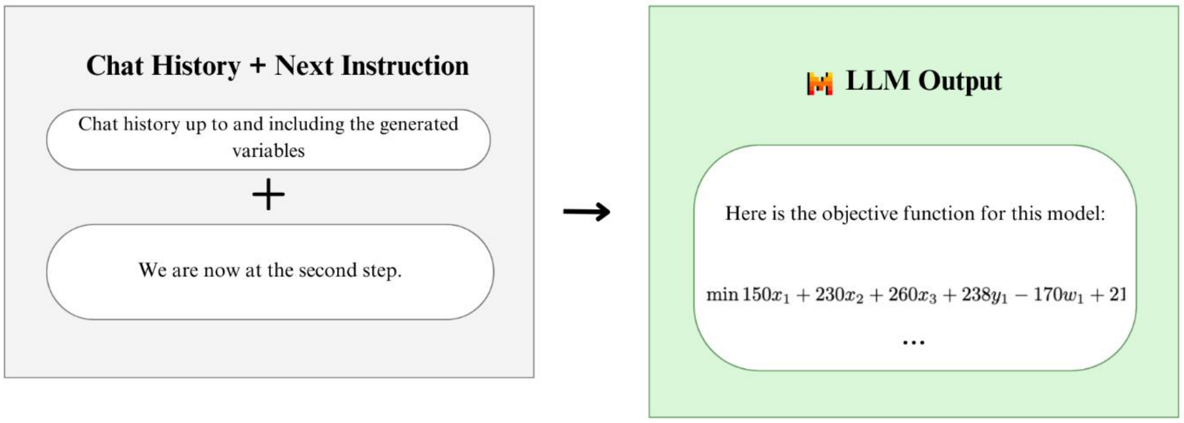
5. *"We are now at the third step."*
   + LLM's response to prompt 3
   + LLM's response to prompt 4

Such division of work resulted in smaller subtasks which could be easier for the LLM to solve, leading to an improved 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.


Prompt + Problem Description + First Instruction
Structural instrucitons
Consider a farmer who specializes in raising wheat, corn, and sugar beets on her 500 acres of land. During the winter, she wants to decide how much land to devote to each crop....
We are now at the first step.
LLM Output

Here are the variables for this model:

x_1 acres of land devoted to wheat
x_2 acres of land devoted to corn
x_3 acres of land devoted to sugar beets
w_1 tons of wheat sold
w_2 tons of corn sold
w_3 tons of sugar beets sold at a favorable price
w_4 tons of sugar beets sold at the lower price
y_1 tons of wheat purchased
y_2 tons of corn purchased


Chat History + Next Instruction
Chat history up to and including the generated variables
+
We are now at the second step.
LLM Output
Here is the objective function for this model:
$\min 150x_1 + 230x_2 + 260x_3 + 238y_1 - 170w_1 + 21$
...

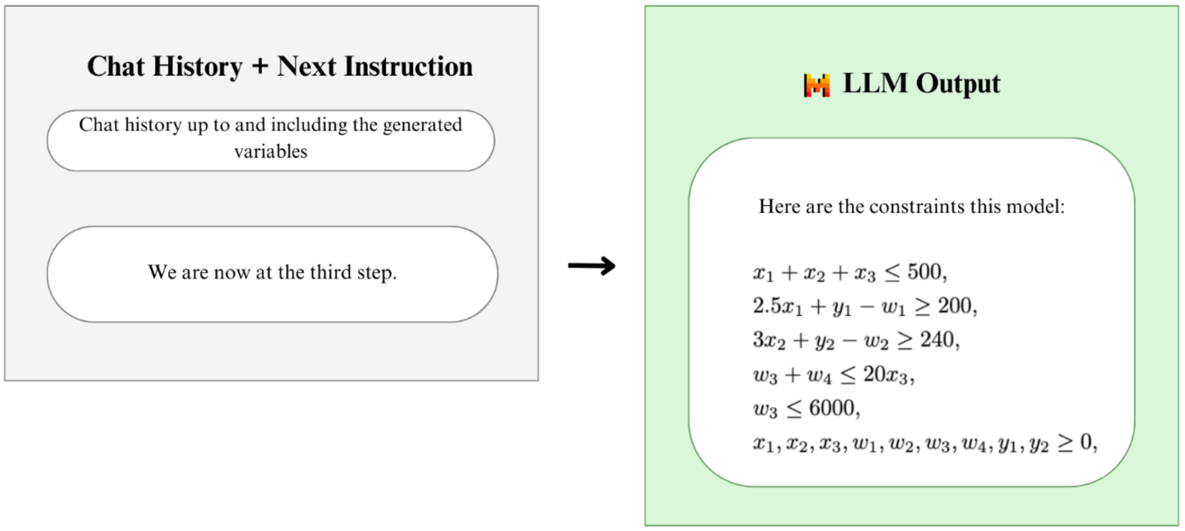
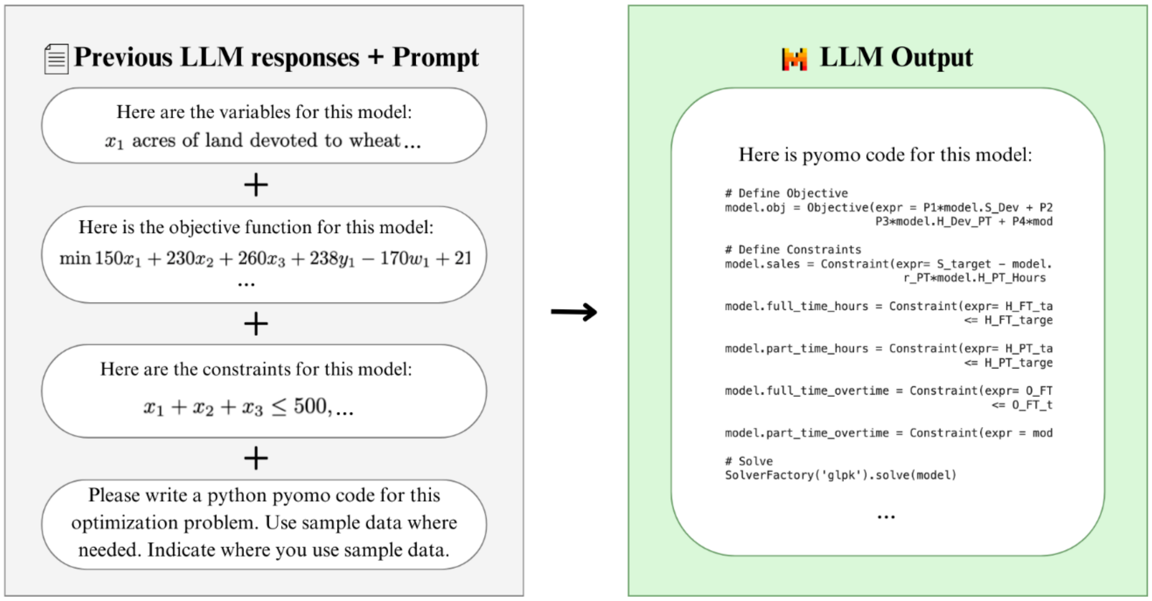

Chat History + Next Instruction
Chat history up to and including the generated variables
We are now at the third step.
LLM Output
Here are the constraints this model:
$x_1 + x_2 + x_3 \leq 500,$
$2.5x_1 + y_1 - w_1 \geq 200,$
$3x_2 + y_2 - w_2 \geq 240,$
$w_3 + w_4 \leq 20x_3,$
$w_3 \leq 6000,$
$x_1, x_2, x_3, w_1, w_2, w_3, w_4, y_1, y_2 \geq 0,$

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 4. An overview of the pipelines is also shown in Figure 5. Next, the generated outputs were evaluated with the chosen metrics.


Previous LLM responses + Prompt
Here are the variables for this model:
x_1 acres of land devoted to wheat ...
+
Here is the objective function for this model:
$\min 150x_1 + 230x_2 + 260x_3 + 238y_1 - 170w_1 + 21$
...
+
Here are the constraints for this model:
$x_1 + x_2 + x_3 \leq 500, \ldots$
+
Please write a python pyomo code for this optimization problem. Use sample data where needed. Indicate where you use sample data.
LLM Output
Here is pyomo code for this model:

```
# Define Objective
model.obj = Objective(expr = P1*model.S_Dev + P2
                      P3*model.H_Dev_PT + P4*mod

# Define Constraints
model.sales = Constraint(expr= S_target - model.
                          r_PT*model.H_PT_Hours

model.full_time_hours = Constraint(expr= H_FT_ta
                                      <= H_FT_targe

model.part_time_hours = Constraint(expr= H_PT_ta
                                      <= H_PT_targe

model.full_time_overtime = Constraint(expr= O_FT
                                          <= O_FT_t

model.part_time_overtime = Constraint(expr = mod

# Solve
SolverFactory('glpk').solve(model)

                    ...
```

Figure 5: The Coding Step of the Experiment Pipeline 2.

3.3 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 (MM) and an evaluation of the code (MC) 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 the concepts of semantics and syntactic transformations of mathematical models which are elementary to metrics used in this study. The semantics of two mathematical models are 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. Changing the syntax of a mathematical problem while preserving its semantics is called a syntactic transformation. Thus, the problem description, reference mathematical formulation and reference code for a given problem are semantically equal but syntactically different. Resulting from this, the measurement of our metrics was conducted under the principle of analyzing and comparing semantics between outputs and references. Examples of this are text-based formulations of constraints in the generated mathematical formulation or wrongly referenced Python functions in code that are still considered to be correct if the semantics are equal to the reference.

3.3.1 Correctness of Mathematical Model (MM1)

This metric represents the LLM's ability to produce a fully correct mathematical representation of the underlying problem (Li et al., 2023; 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:

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

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

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

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

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

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

3.3.2 Objective Value 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 an 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

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

It is important to note that in cases where the generated mathematical model is completely unrepresentative of the of the actual mathematical model according to human judgement, the calculation of the optimal value for the model was omitted and just set to 0. In addition, the feasibility or closeness to the actual optimal solution of a generated mathematical model was not measured. This is because these values highly depend on the parameter values chosen for a specific instance and thus do not provide insights about the performance of the LLM.

3.3.3 Solution Consistency (MM3)

In addition to the accuracy of the generated mathematical model, the consistency of the outputs was of interest. 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:

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

where $n = 3$ is the number of trials executed for a given problem, M is the set of generated mathematical models for a given problem, and $I(M_i = M_j)$ is an indicator function for the semantic equivalence between generated solutions.

3.3.4 Equivalent Code Representation of Mathematical Model (MC1)

This metric represents the LLM's ability to apply a syntactic transformation to its generated mathematical formulation. This metric 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
.

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

3.3.5 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 and was formally defined as:

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

Sometimes, the LLM produces code for printing solution values after solving the model such as the objective value of variable values. These code lines are not considered by this metric.

3.3.6 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 generated sample data was manually replaced by the data associated with the problem. The formula of this metric is:

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

Once again, solution distance and feasibility are not measured for the reasons mentioned above.

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.

4. Results

4.1 Mathematical Model

Following is a description and evaluation of the measurements.

Problem	Parameter Correctness			Variable Correctness			Objective Function Correctness			Constraint Correctness		
	Problem	Class	Dataset	Problem	Class	Dataset	Problem	Class	Dataset	Problem	Class	Dataset
IP1	100.00%	100.00%	85.42%	66.67%	41.67%	37.50%	100.00%	83.33%	52.08%	33.33%	8.33%	8.33%
IP2	100.00%			0.00%			100.00%			0.00%		
IP3	100.00%			100.00%			100.00%			0.00%		
IP4	100.00%			0.00%			33.33%			0.00%		
LP1	100.00%	75.00%		66.67%	41.67%		100.00%	25.00%		0.00%	16.67%	
LP2	0.00%			0.00%			0.00%			0.00%		
LP3	100.00%			100.00%			0.00%			66.67%		
LP4	100.00%			0.00%			0.00%			0.00%		
MIP1	100.00%	91.67%		0.00%	0.00%		100.00%	50.00%		0.00%	0.00%	
MIP2	100.00%			0.00%			0.00%			0.00%		
MIP3	66.67%			0.00%			0.00%			0.00%		
MIP4	100.00%			0.00%			100.00%			0.00%		
NLP1	100.00%	75.00%		100.00%	66.67%		0.00%	50.00%		33.33%	8.33%	
NLP2	33.33%			100.00%			100.00%			0.00%		
NLP3	100.00%			0.00%			33.33%			0.00%		
NLP4	66.67%			66.67%			66.67%			0.00%		

Table 2: Results of Parameter Correctness (MM1P), Variable Correctness (MM1V), Objective Function Correctness (MM1O) and Constraint Correctness (MM1C) for Pipeline 1.

Problem	Parameter Correctness			Variable Correctness			Objective Function Correctness			Constraint Correctness		
	Problem	Class	Dataset	Problem	Class	Dataset	Problem	Class	Dataset	Problem	Class	Dataset
IP1	66.67%	66.67%	72.92%	66.67%	41.67%	54.17%	100.00%	75.00%	58.33%	100.00%	58.33%	22.92%
IP2	0.00%			0.00%			100.00%			33.33%		
IP3	100.00%			100.00%			100.00%			100.00%		
IP4	100.00%			0.00%			0.00%			0.00%		
LP1	100.00%	66.67%		100.00%	66.67%		100.00%	75.00%		33.33%	25.00%	
LP2	0.00%			100.00%			100.00%			0.00%		
LP3	100.00%			66.67%			100.00%			66.67%		
LP4	66.67%			0.00%			0.00%			0.00%		
MIP1	100.00%	100.00%		0.00%	33.34%		66.67%	50.00%		0.00%	0.00%	
MIP2	100.00%			0.00%			0.00%			0.00%		
MIP3	100.00%			66.67%			66.67%			0.00%		
MIP4	100.00%			66.67%			66.67%			0.00%		
NLP1	100.00%	58.33%		100.00%	75.00%		0.00%	33.33%		33.33%	8.33%	
NLP2	33.33%			100.00%			100.00%			0.00%		
NLP3	66.67%			0.00%			0.00%			0.00%		
NLP4	33.33%			100.00%			33.33%			0.00%		

Table 3: Results of Parameter Correctness (MM1P), Variable Correctness (MM1V), Objective Function Correctness (MM1O) and Constraint Correctness (MM1C) for Pipeline 2.

Mixtral 8x22B was able to correctly identify the parameters in 88.89% of problem runs with Pipeline 1 and 72.92% in Pipeline 2. Interestingly, it was not able to identify the parameters of LP2 at all as the problem description included instructions on how to calculate parameters based on some data which the LLM did not follow. Consequently, it could not derive the correct parameters of this problem. Otherwise, the parameters were usually clearly provided and distinguished in the problem description which implies that the LLM was just required to perform a translation from natural language to model component.

The accuracy of the variable identification was roughly equal for IP, LP and NLP problems with better performance in Pipeline 2. With Pipeline 1, variables were correctly identified on at least one trial for 7 problems while this number increased to 10 with Pipeline 2. However, Mixtral was unreliable with the identification of variable domains in problems with variables constraint to different domains. In some cases, such as IP4, the domain of the variables was not explicitly stated but rather implied by its meaning. Hence, the LLM was tasked with deriving the domain through reasoning. However, Mixtral was commonly incorrect and imprecise with such guesses. Hallucinations of variables were rare while missing variables were common. In fact, most of the missed variables were binary indicator variables. Both pipelines performed worst on MIP problems which commonly include indicator variables. IP1 and IP3 belong to the assignment problem domain and outperformed other domains within the IP problems.

The accuracy of objective function correctness and constraint correctness is heavily dependent on the accuracy of parameter and variable identification as it uses those as components to construct the objective function. However, in the case of IP2, IP4, LP1, MIP1 and MIP4, a correct objective function was still achieved by the LLM on some runs despite prior identifying variables or parameters incorrectly. In all cases, the wrong variables were used in such a way that the semantic intention of the objective function was still captured. Contrarily, correct variables were represented correctly in the objective function in almost all cases.

Mixtral's performance on the identification of constraints was worse compared to other model components. Notably, this metric is applied to a group of constraints where each constraint can be a source of error. Thus, the low performance can be partially attributed to the larger number of error sources that are covered by this metric. The commonly missed indicator variables usually led to a complete misrepresentation of the problem. This is because indicator variables enable more complex model structure which is representative of the business-level problems in the dataset. Besides, other errors found in wrong constraints varied and were usually problem-specific. Usually, a specific model component that was mentioned but not described in detail led to a misinterpretation by the LLM. As an example, MIP4 is a network flow problem with a temporal component. In addition, flow can be released from the system at all nodes to ensure that edges are not overflowing. Here, Mixtral did not understand this mechanic and thus either reverted to omitting model components related to this mechanic or simply improvised. Interestingly, constraints with this background were wrong across multiple trials but were inconsistent in how Mixtral modeled them.

Problem	Model Correctness			Objective Value Optimality			Solution Consistency		
	Problem	Class	Dataset	Problem	Class	Dataset	Problem	Class	Dataset
IP1	33.33%	8.33%	2.08%	33.33%	33.33%	16.67%	0.00%	8.33%	12.50%
IP2	0.00%			0.00%			0.00%		
IP3	0.00%			100.00%			33.33%		
IP4	0.00%			0.00%			0.00%		
LP1	0.00%	0.00%		0.00%	0.00%		0.00%	8.33%	
LP2	0.00%			0.00%			0.00%		
LP3	0.00%			0.00%			0.00%		
LP4	0.00%			0.00%			33.33%		
MIP1	0.00%	0.00%		0.00%	0.00%		100.00%	25.00%	
MIP2	0.00%			0.00%			0.00%		
MIP3	0.00%			0.00%			0.00%		
MIP4	0.00%			0.00%			0.00%		
NLP1	0.00%	0.00%		100.00%	33.33%		33.33%	8.33%	
NLP2	0.00%			33.33%			0.00%		
NLP3	0.00%			0.00%			0.00%		
NLP4	0.00%			0.00%			0.00%		

Table 4 Results of Model Correctness (MM1), Objective Value Optimality (MM2) and Solution Consistency (MM3) for Pipeline 1.

Problem	Model Correctness			Objective Value Optimality			Solution Consistency		
	Problem	Class	Dataset	Problem	Class	Dataset	Problem	Class	Dataset
IP1	66.67%	41.67%	14.58%	66.67%	41.67%	22.92%	0.00%	0.00%	2.08%
IP2	0.00%			0.00%			0.00%		
IP3	100.00%			100.00%			0.00%		
IP4	0.00%			0.00%			0.00%		

LP1	33.33%	16.67%		66.67%	25.00%		0.00%	0.00%	
LP2	0.00%			0.00%			0.00%		
LP3	33.33%			33.33%			0.00%		
LP4	0.00%			0.00%			0.00%		
MIP1	0.00%	0.00%		0.00%	0.00%		0.00%	0.00%	
MIP2	0.00%			0.00%			0.00%		
MIP3	0.00%			0.00%			0.00%		
MIP4	0.00%			0.00%			0.00%		
NLP1	0.00%	0.00%		66.67%	25.00%		0.00%	8.33%	
NLP2	0.00%			33.33%			33.33%		
NLP3	0.00%			0.00%			0.00%		
NLP4	0.00%			0.00%			0.00%		

Table 5: Results of Model Correctness (MM1), Objective Value Optimality (MM2) and Solution Consistency (MM3) for Pipeline 2.

As a result of incorrect parameters, variables, objective function, and constraints the LLM had a performance of 2.78% on model correctness with Pipeline 1 and 14.58% with Pipeline 2, where most errors were caused by missed or wrong constraints. Interestingly, on some trials such as all trials of IP3 of Pipeline 1, faulty mathematical models returned an objective value that is equal to the optimal objective value of the reference model. This may be attributed to minor alterations of the generated mathematical models which still led to optimal objective values. However, it is important to note that this is mainly due to the coincidence that the specific problem instance with its parameter values was insensitive to these alterations. Pipeline 1 was only able to produce correct mathematical models for IP problems while Pipeline 2 was able to increase performance on IP problems and generate correct LP models. Once again, problems belonging to the assignment problem domain outperformed other problem domains in terms of objective value optimality.

Generally, the LLM was inconsistent with the mathematical formulations generated across several trials. Usually, a difference in generated constraints caused the model to be inconsistent but some inconsistencies were attributed to objective functions and variables. Across all trials of Pipeline 2, only two runs were semantically equivalent which is lower than in Pipeline 1. In addition, the responses were a lot more verbose and had more detailed explanations of the model.

4.2 Code

Problem	**Equivalence to Model**			**Code Executability**			**Code Solution Optimality**		
	Problem	**Class**	**Dataset**	**Problem**	**Class**	**Dataset**	**Problem**	**Class**	**Dataset**
IP1	66.67%	66.67%	83.33%	33.33%	50.00%	41.67%	0.00%	25.00%	16.67%
IP2	33.33%			0.00%			0.00%		
IP3	100.00%			66.67%			100.00%		
IP4	66.67%			100.00%			0.00%		
LP1	100.00%	91.67%		66.67%	33.34%		33.33%	8.33%	
LP2	100.00%			0.00%			0.00%		

LP3	66.67%			66.67%			0.00%		
LP4	100.00%			0.00%			0.00%		
MIP1	100.00%	75.00%		66.67%	25.00%		0.00%	0.00%	
MIP2	100.00%			0.00%			0.00%		
MIP3	66.67%			0.00%			0.00%		
MIP4	33.33%			33.33%			0.00%		
NLP1	100.00%	100.00%		100.00%	58.34%		100.00%	33.33%	
NLP2	100.00%			66.67%			33.33%		
NLP3	100.00%			0.00%			0.00%		
NLP4	100.00%			66.67%			0.00%		

Table 6: Results of Code Equivalence to Mathematical Model (MC1), Code Executability (MC2) and Code Solution Optimality (MC3) for Pipeline 1.

Problem	**Equivalence to Model**			**Code Executability**			**Code Solution Optimality**		
	Problem	**Class**	**Dataset**	**Problem**	**Class**	**Dataset**	**Problem**	**Class**	**Dataset**
IP1	33.33%	75.00%	83.33%	66.67%	16.67%	27.08%	33.33%	33.33%	20.83%
IP2	100.00%			0.00%			0.00%		
IP3	100.00%			0.00%			100.00%		
IP4	66.67%			0.00%			0.00%		
LP1	100.00%	83.33%		100.00%	50.00%		66.67%	25.00%	
LP2	33.33%			33.33%			0.00%		
LP3	100.00%			66.67%			33.33%		
LP4	100.00%			0.00%			0.00%		
MIP1	100.00%	91.67%		66.67%	16.67%		0.00%	0.00%	
MIP2	100.00%			0.00%			0.00%		
MIP3	100.00%			0.00%			0.00%		
MIP4	66.67%			0.00%			0.00%		
NLP1	100.00%	83.34%		66.67%	25.00%		66.67%	25.00%	
NLP2	100.00%			0.00%			33.33%		
NLP3	66.67%			0.00%			0.00%		
NLP4	66.67%			33.33%			0.00%		

Table 7: Results of Code Equivalence to Mathematical Model (MC1), Code Executability (MC2) and Code Solution Optimality (MC3) for Pipeline 2.

The model equivalence was 83.33% across all problems and was similar across all problem classes for both pipelines. For each generated code, a majority of the model components were semantically equivalent between mathematical formulation and code. Errors were usually very minor and located in highly nested mathematical expressions such as chained multiplication or piece-wise functions. Model components in the mathematical model always had a counterpart in the code. New components have not been hallucinated. Thus, Mixtral was able to identify and distinguish between model components and convert them into an accurate code representation for the most part.

The executability of the generated code decreased from 41.67% in Pipeline 1 to 27.08% in Pipeline 2. The code errors can mainly be attributed to an unfamiliarity with the Pyomo package of Mixtral 8x22B. Common errors included hallucination of Pyomo functions, wrong usage of existing Pyomo functions, modelling logic that was not supported by Pyomo and wrong indexation of parameters and variables. In addition, Pipeline 2 commonly combined multiple constraints into a single constraint.

With Pipeline 1, 4 out of 16 problems were solved correctly on at least one trial while this number increased to 6 with Pipeline 2. Mainly, simple IP and LP problems were solved fully autonomously. Interestingly, the code solution correctness is higher than the model correctness. Once again, this is due to small model errors that did not lead to a change in the optimality of the solution.

In summary, Mixtral 8x22B solved only 16.67% of the problems optimally with Pipeline 1 and 20.83% with Pipeline 2. Little to no trend was visible in terms of problem domains. While Assignment problems performed slightly better on some metrics compared to other domains, this may be attributed to their level of complexity and similarity to textbook problems. The LLM performed best on the generation of mathematical models for IP and LP problems which may also be attributed to their level of complexity. The LLM struggled with model generation for MIP problems due to their common usage of indicator variables. While generating the mathematical model, the most common source of error was missing or incorrect constraints. In particular, the LLM struggled to generate indicator variables and constraints related to indicator variables. In addition, the LLM struggled with the correct application of real-world concepts and mechanisms that seem trivial to a human expert when the mechanisms were not specifically stated in the problem description. When simply asked to convert the mathematical formulation into code an accuracy of 83.33% was achieved with both pipelines. This shows that Mixtral 8x22B has a high native capability of understanding input and keeping semantics when provided with enough information. This capability suffers when input becomes implied or ambiguous as it forces the LLM to make incorrect guesses.

4.3 Comparison between Pipeline 1 and Pipeline 2

Despite the seemingly better performance of Pipeline 2 on model generation metrics and solution optimality, there is no statistical evidence for differences in any of the metrics[1]. Thus, the signal for better performance with Pipeline 2 is weak. Because the mathematical model was generated through multiple steps in Pipeline 2, it introduced the possibility of self-correcting behavior as the LLM could possibly identify missed or wrong model components of previous steps. However, this only occurred once during a trial of IP2 where the LLM introduced an indicator variable while generating the constraints but did not lead to a correct set of constraints. In addition, the mathematical models of Pipeline 2 were more verbose as more tokens were used to explain each model component more thoroughly. This led to an easier assessment of the model's correctness and improved interpretability of the mathematical model. However, because the pipelines generated the model step by step in the order of parameters, variables, objective function and constraints, it was not able to understand what current model components would be required to accurately generate consecutive model components. Generating the model step by step in Pipeline 2 did not solve this problem.

[1] P-values were calculated through a two-sided t-test with a significance level of 0.05.

5. Deliverable

The dataset, all trials and measurements may be found on this GitHub page:

https://github.com/RiVuss/LLMsForOptimizationModelling

The repository includes similar research on the LLMs GPT-4 (Melis Cevik), Gemini 1.5 Pro (Hubert Perlinski) and Claude 3 (Annika Siefke).

6. Conclusion

Overall, the percentage of correct solutions produced by Mixtral is insufficient to consider the combination of Pipeline 1 or Pipeline 2 with Mixtral as an autonomous optimization solver for business applications as only 16.67% of problems were solved correctly with pipeline and 20.83% with Pipeline 2. Given the lack of statistical evidence for differences in performance between the two pipelines, other pipelines should be considered to increase the optimality of results achieved with Mixtral. In general, the generation of the mathematical model may be seen as the most critical step as the LLM oftentimes failed to generate a correct model as a first step. The most common error in the model generation step is a misunderstanding of unwritten and implied model components such as variable domains and indicator variables. These misunderstandings usually led to further errors in objective functions and constraints as the underlying mechanisms were not modeled in a correct mathematical way.Contrarily, Mixtral demonstrated the ability to declare parameters correctly in most trials. This is because parameters were usually given as a clear and distinguishable model component. Thus, the translation from natural language to model component was more reliable for parameters. Furthermore, the models generated by Mixtral were inconsistent, as the consistency was measured at 12.50% for Pipeline 1 and 2.08% for Pipeline 2. Generally, correct model components remained correct across trials while previously incorrect model components were modeled inconsistently wrong. Errors related to the code generation were mainly based on an improper usage of the Pyomo library. A high equivalence score of 83.33% for both pipelines once again indicates that Mixtral was able to reproduce clearly stated information, similar to the parameter identification.

These findings contradict the findings of Xiao et al. (2023) who found an improvement in the accuracy of the generated mathematical models after breaking the generation step down into multiple subprocesses. Given the lack of statistical evidence for differences in performances on all metrics between the pipelines in this research, no performance improvement was found when utilizing *Chain of Thought* prompting. Gangwar & Kani (2023) found that the utilization of subproblems in the model generation did not lead to more accurate mathematical models. Thus, it remains uncertain under which conditions solving subproblems may lead to more accurate optimization models generated by LLMs.

The suggestion that unwritten and implied model components in the problem descriptions lead to misinterpretations by the LLM suggests that inconsistent problem descriptions may be problematic for the model generation step. This finding is similar to a finding of AhmadiTeshnizi et al. (2023) who utilized a schema for their problem descriptions to reduce inconsistencies in problem descriptions. While a fixed problem description schema may limit the usability of an autonomous optimization software, it may lead to more accurate models.

Due to the limited number of trials that led to an optimal model or solution, it is suggested that Mixtral is not capable of performing the role of a reliable autonomous optimization solver, which supports the findings of Yan et al. (2023) who claim that LLMs are currently not able to perform reliably in this task

7. Limitations

Several limitations apply to this research. First, the created dataset of 16 business-level problems is too small of a dataset to properly generalize the results on the utilization of LLMs as business-level optimizers. Business applications are very broad and unique to each use case. In addition, the scale of optimization in real applications has a high variation. While this dataset is diverse in terms of domains and difficulty, it is not close to being representative of the variation of problems found in real applications. Second, the pipelines used in this research are very simple and do not reach the complexity of the pipelines utilized by researchers mentioned. As mentioned, complex pipelines have been shown to increase the accuracy of the outputs. Thus, it is difficult to assess the true potential of Mixtral 8x22B purely based on this study.

8. Future Research

Based on the results and conclusions stated above, several future research is suggested. First, this research showed that imprecise problem descriptions led to a confused and guessing Mixtral. Thus, future research should investigate how possible pipelines can be adjusted to increase the robustness towards unstructured problem descriptions. Second, the low usage of indicator variables in the generated mathematical models suggests an unfamiliarity with more complex modeling techniques. Hence, research on how to familiarize and teach LLMs to use more complex modeling techniques is of essence as they will be eventually required for solving business applications. Third, mistakes in early stages of the pipeline generally propagated throughout consecutive steps. Thus, it may be valuable to investigate how mistakes can be investigated more autonomously. This research finds that wrong model components tend to stay wrong across multiple trials. However, they tend to be wrong in inconsistent ways which may serve as an entry point for future research. Finally, the pipelines assume that feeding the parameter values into the code can be solved through efforts in software engineering without specifying what these efforts are. Thus, an investigation on this is also an elementary step towards an autonomous LLM optimizer.

9. Statement on Work and Originality

This thesis includes both individual and collaborative elements. The following (sub)sections are predominantly based on group work:

- Section 3.1 Data Description
- Section 3.2 Experimental Pipelines
- Section 3.3 Metrics

All the other sections in this thesis are my individual contributions. I, Nathan Ferchtandiker, take full responsibility for all work presented in this thesis irrespective of it being based 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.

10. Acknowledgement

I would like to sincerely thank Hubert Perlinski, Annika Siefke and Melis Cevik for their contributions to this research. This research was conducted in collaboration as we equally contributed towards the design of the methodology. This includes close cooperation on the selection of problem classes, creation of the dataset, design of the pipelines and definition of the metrics. All members were well organized and cooperated with a good research outcome in mind. In addition, I would like to thank Donato Maragno for the supervision of the thesis as he gave us valuable resources that served as a gentle introduction into this research and continued to give us guidance throughout this project while providing us with the freedom to design this research according to our preferences.

11. 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

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

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*.

Anthropic. (2024, March 7). *Introducing the next generation of claude. Introducing the next generation of Claude.* https://www.anthropic.com/news/claude-3-family

Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Besta, M., Memedi, F., Zhang, Z., Gerstenberger, R., Blach, N., Nyczyk, P., ... & Hoefler, T. (2024). Topologies of Reasoning: Demystifying Chains, Trees, and Graphs of Thoughts. *arXiv preprint arXiv:2401.14295*.

Beltagy, I., Lo, K., & Cohan, A. (2019). SciBERT: A pretrained language model for scientific text. *arXiv preprint arXiv:1903.10676*.

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. https://apps.dtic.mil/sti/pdfs/AD0679037.pdf

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., ... & Amodei, D. (2020). Language models are few-shot learners. *Advances in neural information processing systems*, *33*, 1877-1901.

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

Chronopoulou, A., Baziotis, C., & Potamianos, A. (2019). An embarrassingly simple approach for transfer learning from pretrained language models. *arXiv preprint arXiv:1902.10547*.

Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., ... & Schulman, J. (2021). Training verifiers to solve math word problems, 2021. *URL https://arxiv. org/abs/2110.14168*.

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

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.

Dong, Q., Li, L., Dai, D., Zheng, C., Wu, Z., Chang, B., ... & Sui, Z. (2022). A survey on in-context learning. *arXiv preprint arXiv:2301.00234*.

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*.

Gangwar, N., & Kani, N. (2023, August). Highlighting named entities in input for auto-formulation of optimization problems. In *International Conference on Intelligent Computer Mathematics* (pp. 130-141). Cham: Springer Nature Switzerland.

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

Hendrycks, D., Burns, C., Basart, S., Zou, A., Mazeika, M., Song, D., & Steinhardt, J. (2020). Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*.

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

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., ... & Chen, W. (2021). Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.

Jiang, Albert Q., Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand et al. "Mistral 7B." *arXiv preprint arXiv:2310.06825* (2023).

Jiang, A. Q., Sablayrolles, A., Roux, A., Mensch, A., Savary, B., Bamford, C., ... & Sayed, W. E. (2024). Mixtral of experts. *arXiv preprint arXiv:2401.04088*.

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., ... & Kiela, D. (2020). Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, *33*, 9459-9474.

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

Microsoft. (2024, January 30). *Getting started with LLM Fine-Tuning*. Microsoft Learn. https://learn.microsoft.com/en-us/ai/playbook/technology-guidance/generative-ai/working-with-llms/fine-tuning

Minaee, S., Mikolov, T., Nikzad, N., Chenaghlu, M., Socher, R., Amatriain, X., & Gao, J. (2024). Large language models: A survey. *arXiv preprint arXiv:2402.06196*.

Mistral AI. (2024, April 17). *Cheaper, better, faster, stronger*. Mistral AI | Frontier AI in your hands. https://mistral.ai/news/mixtral-8x22b/

Mizrahi, M., Kaplan, G., Malkin, D., Dror, R., Shahaf, D., & Stanovsky, G. (2023). State of what art? a call for multi-prompt llm evaluation. *arXiv preprint arXiv:2401.00595*.

MOSEK. (n.d.). *11.3 robust linear optimization — mosek optimization toolbox for matlab* 10.2.0. Retrieved May 23, 2024, from https://docs.mosek.com/latest/toolb ox/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

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., ... & Lowe, R. (2022). Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, *35*, 27730-27744.

Pedroso, J. P., Rais, A., Kubo, M., & Muramatsu, M. (n.d.). *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.* In Springer eBooks. https://doi.org/10.1007/978-1-4471-5577-5

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

Ramamonjison, R., Yu, T., Xing, L., Mostajabdaveh, M., Li, X., Fu, X., ... & Zhang, Y. (2023a, July). LaTeX2Solver: a Hierarchical Semantic Parsing of LaTeX Document into Code for an Assistive Optimization Modeling Application. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)* (pp. 471-478).

Ramamonjison, R., Yu, T., Li, R., Li, H., Carenini, G., Ghaddar, B., ... & Zhang, Y. (2023b, August). NL4Opt competition: Formulating optimization problems based on their natural language descriptions. In *NeurIPS 2022 Competition Track* (pp. 189-203). PMLR.

Sarker, L., Downing, M., Desai, A., & Bultan, T. (2024, April 1). Syntactic robustness for LLM-based code generation. *arXiv.org*. https://arxiv.org/abs/2404.01535

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

Stok, D. (2022). *Optimization truck loading and scheduling. A thesis for PostNL* [Master's thesis, Erasmus University Rotterdam]. Erasmus University Thesis Repository. hdl.handle.net/2105/62818

Team, G., Anil, R., Borgeaud, S., Wu, Y., Alayrac, J. B., Yu, J., ... & Ahn, J. (2023). *Gemini: a family of highly capable multimodal models*. *arXiv preprint arXiv:2312.11805*.

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M. A., Lacroix, T., ... & Lample, G. (2023). *Llama: Open and efficient foundation language models*. *arXiv preprint arXiv:2302.13971*.

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

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

Xiao, Z., Zhang, D., Wu, Y., Xu, L., Wang, Y. J., Han, X., ... & Chen, G. (2023, October). Chain-of-Experts: When LLMs Meet Complex Operations Research Problems. In *The Twelfth International Conference on Learning Representations*.

Yang, C., Wang, X., Lu, Y., Liu, H., Le, Q. V., Zhou, D., & Chen, X. (2023). Large language models as optimizers. *arXiv preprint arXiv:2309.03409*.

Zak, E. J. (2024). *How to solve real-world optimization Problems.* In SpringerBriefs in operations research. https://doi.org/10.1007/978-3-031-49838-1

12. Appendix

12.1 Example Problem

12.1.1 Description

You are tasked with scheduling the power output of 6 electric power thermal units over the timespan of 15 periods. There is a constant startup cost for each power unit that is applied if the power plant is turned on. The startup costs are 10324€, 5678€, 7802€, 12899€, 4596€ and 9076€ for powerplants 1 to 6, respectively. In addition, there is a constant shutdown cost for each power unit that is applied if the power plant is turned off. The shutdown costs are 2673€, 5893€, 982€, 6783€, 2596€ and 3561€ for powerplants 1 to 6, respectively. There is also a fixed and variable cost applied if the power plant is running. The fixed cost is constant and the variable cost is proportional to the output of a power plant. There are lower and upper bounds for the output power for each unit. The variable costs are different for each power plant and at each time step. Next, there are maximum power increments and decrements for each power plant that limit how much the output can change from one time period to the next. There is a total power demand that needs to be fulfilled by the power units. Finally, for security reasons, the total available power out should always be 10% higher than the demand.

Mathematical formulation

I: Set of power plants
T: Set of time periods

$X_{i,t}$: output of plant i at time t
$Y_{i,t}$: 1 if plant i starts up at time t
$V_{i,t}$: 1 if plant i starts up or continues to run at time t
$Z_{i,t}$: 1 if plant i shuts down at time t

crf_i: fixed cost of running plant i
crf_i: variable cost of running plant i
cu_i: cost associated with starting a plant up
cd_i: cost associated with shutting a plant down
l_i: lower bound for output of plant i
u_i: upper bound for output of plant i
s_i: maximum power ramp up between periods of plant i
t_i: maximum power ramp down between periods of plant i
$x_i^0 = 0 \quad \forall i \in I$: initial power output of plant i
$v_i^0 = 0 \quad \forall i \in I$: initial running state of plant i
d_t: power demand at time t

$$
\begin{array}{lrll}
\text{Minimize} & \multicolumn{3}{l}{\sum_{i\in I}\sum_{t\in T} crf_i V_{i,t} + crv_i X_{i,t} + cu_i Y_{i,t} + cd_i Z_{i,t}} \\
\text{Subject to:} & l_i V_{i,t} & \leq X_{i,t}, & \forall i \in I, t \in T \\
 & u_i V_{i,t} & \geq X_{i,t}, & \forall i \in I, t \in T \\
 & X_{i,t+1} - X_{i,t} & \leq s_i, & \forall i \in I, t \in T - \{1\} \\
 & X_{i,1} - x_i^0 & \leq s_i, & \forall i \in I \\
 & X_{i,t} - X_{i,t+} & \leq t_i, & \forall i \in I, t \in T - \{1\} \\
 & x_i^0 - X_{i,1} & \leq t_i, & \forall i \in I \\
 & Y_{i,t} - Z_{i,t} & = V_{i,t} - V_{i,t-1}, & \forall i \in I, t \in T - \{1\} \\
 & Y_{i,1} - Z_{i,1} & = V_{i,1} - v_i^0, & \forall i \in I \\
 & \sum_{i\in I} X_{i,t} & = d_t, & \forall t \in T \\
 & \sum_{i\in I} u_i V_{i,t} & = 1.1 d_t, & \forall t \in T \\
 & X_{i,t} & \in R^+ & \forall i \in I, t \in T \\
 & Y_{i,t} & \in \{0,1\} & \forall i \in I, t \in T \\
 & Z_{i,t} & \in \{0,1\} & \forall i \in I, t \in T \\
 & Y_{i,t} & \in \{0,1\} & \forall i \in I, t \in T
\end{array}
$$

12.1.2 Interaction with Mixtral 8x22B on Pipeline 1

12.1.2.1 Mathematical Model

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

User
You are tasked with scheduling the power output of 6 electric power thermal units over the timespan of 15 periods. There is a constant startup cost for each power unit that is applied if the power plant is turned on. The startup costs are 10324€, 5678€, 7802€, 12899€, 4596€ and 9076€ for powerplants 1 to 6, respectively. In addition, there is a constant shutdown cost for each power unit that is applied if the power plant is turned off. The shutdown costs are 2673€, 5893€, 982€, 6783€, 2596€ and 3561€ for powerplants 1 to 6, respectively. There is also a fixed and variable cost applied if the power plant is running. The fixed cost is constant and the variable cost is proportional to the output of a power plant. There are lower and upper bounds for the output power for each unit. The variable costs are different for each power plant and at each time step. Next, there are maximum power increments and decrements for each power plant that limit how

much the output can change from one time period to the next. There is a total power demand that needs to be fulfilled by the power units. Finally, for security reasons, the total available power out should always be 10% higher than the demand.

Mixtral 8x22B
To formulate a mathematical optimization model for this problem, we start by defining the parameters, decision variables, objective function, and constraints.

Parameters:

Let C1_available = 10,000 l and C2_available = 15,000 l be the available quantities of components C1 and C2, respectively.
Let GCA_demand = 6,000 l, GCB_demand = 7,000 l, and GCC_demand = 9,000 l be the minimum demand levels for products GCA, GCB, and GCC, respectively.
Let C1_critical = 0.4 and C2_critical = 0.2 be the proportional critical elements for components C1 and C2, respectively.
Let GCA_critical_min = 0.3, GCB_critical_max = 0.3, and GCC_ratio_min = 0.3 be the critical element requirements for products GCA, GCB, and GCC, respectively.
Let GCA_profit = $120, GCB_profit = $135, and GCC_profit = $155 be the profits per litre for products GCA, GCB, and GCC, respectively.
Decision variables:

Let x1, x2, and x3 be the quantities of products GCA, GCB, and GCC to be produced, respectively.
Objective function:

The objective is to maximize the total profit, which can be expressed as:

Maximize Z = 120x1 + 135x2 + 155x3

Constraints:

Availability of components:

C1_used = x1 + x2 + x3
C2_used = x1 + x2 + x3
C1_used <= C1_available
C2_used <= C2_available
Minimum demand requirements:

x1 >= GCA_demand
x2 >= GCB_demand
x3 >= GCC_demand
Critical element requirements:

(0.4x1 + 0.2x2) / (x1 + x2) >= GCA_critical_min (for GCA)
(0.4x1 + 0.2x2) / (x1 + x2) <= GCB_critical_max (for GCB)
x1 / x3 >= GCC_ratio_min (for GCC)
Non-negativity constraints:

x1 >= 0
x2 >= 0
x3 >= 0
This formulation aims to maximize the profit while satisfying the resource availability, minimum demand requirements, and critical element requirements. To solve this optimization problem, one can use linear programming techniques or other optimization methods.

12.1.2.2 Code

User
Please write a Python Pyomo code for this optimization problem. Use sample data where needed. Indicate where you use sample data.

User
To formulate a mathematical optimization model for this problem, we start by defining the parameters, decision variables, objective function, and constraints.

User
Parameters:

Let C1_available = 10,000 l and C2_available = 15,000 l be the available quantities of components C1 and C2, respectively.
Let GCA_demand = 6,000 l, GCB_demand = 7,000 l, and GCC_demand = 9,000 l be the minimum demand levels for products GCA, GCB, and GCC, respectively.
Let C1_critical = 0.4 and C2_critical = 0.2 be the proportional critical elements for components C1 and C2, respectively.
Let GCA_critical_min = 0.3, GCB_critical_max = 0.3, and GCC_ratio_min = 0.3 be the critical element requirements for products GCA, GCB, and GCC, respectively.
Let GCA_profit = $120, GCB_profit = $135, and GCC_profit = $155 be the profits per litre for products GCA, GCB, and GCC, respectively.
Decision variables:

Let x1, x2, and x3 be the quantities of products GCA, GCB, and GCC to be produced, respectively.
Objective function:

The objective is to maximize the total profit, which can be expressed as:

Maximize Z = 120x1 + 135x2 + 155x3

Constraints:

Availability of components:

C1_used = x1 + x2 + x3
C2_used = x1 + x2 + x3
C1_used <= C1_available
C2_used <= C2_available
Minimum demand requirements:

x1 >= GCA_demand
x2 >= GCB_demand
x3 >= GCC_demand
Critical element requirements:

(0.4x1 + 0.2x2) / (x1 + x2) >= GCA_critical_min (for GCA)
(0.4x1 + 0.2x2) / (x1 + x2) <= GCB_critical_max (for GCB)
x1 / x3 >= GCC_ratio_min (for GCC)
Non-negativity constraints:

x1 >= 0
x2 >= 0
x3 >= 0
This formulation aims to maximize the profit while satisfying the resource availability, minimum demand requirements, and critical element requirements. To solve this optimization problem, one can use linear programming techniques or other optimization methods.

Mixtral 8x22B
Here is a Python code using Pyomo to solve the optimization problem:

```
# Import the necessary packages
from Pyomo.environ import *

# Define the model
model = ConcreteModel()

# Define the parameters (sample data)
model.C1_available = 10000
model.C2_available = 15000
model.GCA_demand = 6000
model.GCB_demand = 7000
model.GCC_demand = 9000
model.C1_critical = 0.4
model.C2_critical = 0.2
model.GCA_critical_min = 0.3
model.GCB_critical_max = 0.3
model.GCC_ratio_min = 0.3
model.GCA_profit = 120
model.GCB_profit = 135
model.GCC_profit = 155

# Define the decision variables
model.x1 = Var(within=NonNegativeReals) # quantity of GCA
model.x2 = Var(within=NonNegativeReals) # quantity of GCB
model.x3 = Var(within=NonNegativeReals) # quantity of GCC

# Define the objective function
model.obj = Objective(expr=model.GCA_profit * model.x1 + model.GCB_profit * model.x2 +
model.GCC_profit * model.x3, sense=maximize)

# Define the constraints
model.C1_used = Constraint(expr=model.x1 + model.x2 + model.x3 <= model.C1_available)
model.C2_used = Constraint(expr=model.x1 + model.x2 + model.x3 <= model.C2_available)
model.GCA_demand_req = Constraint(expr=model.x1 >= model.GCA_demand)
model.GCB_demand_req = Constraint(expr=model.x2 >= model.GCB_demand)
model.GCC_demand_req = Constraint(expr=model.x3 >= model.GCC_demand)
model.GCA_critical_req = Constraint(expr=(model.C1_critical * model.x1 + model.C2_critical *
model.x2) / (model.x1 + model.x2) >= model.GCA_critical_min)
model.GCB_critical_req = Constraint(expr=(model.C1_critical * model.x1 + model.C2_critical *
model.x2) / (model.x1 + model.x2) <= model.GCB_critical_max)
model.GCC_ratio_req = Constraint(expr=model.x1 / model.x3 >= model.GCC_ratio_min)

# Solve the model
SolverFactory('glpk').solve(model)

# Print the results
print("Optimal profit: ", model.obj())
print("Quantity of GCA: ", model.x1())
print("Quantity of GCB: ", model.x2())
print("Quantity of GCC: ", model.x3())
```
This code defines the optimization model using Pyomo, solves it using the GLPK solver, and prints the optimal solution. Note that the sample data is used for the parameter values as indicated in the problem statement.

12.1.3 Interaction with Mixtral 8x22B on Pipeline 2

12.1.3.1 Mathematical Model

User

Please formulate a mathematical optimization model for this problem.
It is important that you do this by following these steps:

1. Define parameters and variables
2. Define the objective function
3. Define the constraints

Please note that each answer is always dedicated to one step. Therefore, do not provide extra information than what is asked. The answers will be manually combined by the user to create the full model.

Usef
You are tasked with scheduling the power output of 6 electric power thermal units over the timespan of 15 periods. There is a constant startup cost for each power unit that is applied if the power plant is turned on. The startup costs are 10324€, 5678€, 7802€, 12899€, 4596€ and 9076€ for powerplants 1 to 6, respectively. In addition, there is a constant shutdown cost for each power unit that is applied if the power plant is turned off. The shutdown costs are 2673€, 5893€, 982€, 6783€, 2596€ and 3561€ for powerplants 1 to 6, respectively. There is also a fixed and variable cost applied if the power plant is running. The fixed cost is constant and the variable cost is proportional to the output of a power plant. There are lower and upper bounds for the output power for each unit. The variable costs are different for each power plant and at each time step. Next, there are maximum power increments and decrements for each power plant that limit how much the output can change from one time period to the next. There is a total power demand that needs to be fulfilled by the power units. Finally, for security reasons, the total available power out should always be 10% higher than the demand.

User
We are now at the first step.

Mixtral 8x22B
Step 1: Define parameters and variables

Let's denote:
- i as the index for power units, where $i \in \{1,2,3,4,5,6\}$
- t as the index for time periods, where $t \in \{1,2,\ldots,15\}$
- C^{startup}_i as the startup cost for power unit i
- C^{shutdown}_i as the shutdown cost for power unit i
- C^{fixed}_i as the fixed cost for power unit i
- C^{variable}_{it} as the variable cost for power unit i at time t
- P^{min}_i as the minimum output power for power unit i
- P^{max}_i as the maximum output power for power unit i
- Δ^{up}_i as the maximum power increment for power unit i
- Δ^{down}_i as the maximum power decrement for power unit i
- D_t as the power demand at time t
- $y_{it} \in \{0,1\}$ as a binary variable indicating whether power unit i is turned on at time t

- $u_{it} \in [0,1]$ as a continuous variable representing the fraction of the maximum output power of power unit i at time t

The power output of power unit i at time t is given by $P_{it} = P^{\text{max}}_i \cdot u_{it}$.

User
We are now at the second step.

Mixtral 8x22B
Step 2: Define the objective function

The objective function is the total cost, which includes the startup, shutdown, fixed, and variable costs. We want to minimize this total cost. The objective function can be formulated as follows:

\[
\text{minimize} \sum_{t=1}^{15} \left(\sum_{i=1}^{6} C^{\text{startup}}_i \cdot (y_{it} - y_{i,t-1}) + C^{\text{shutdown}}_i \cdot (y_{i,t-1} - y_{it}) + C^{\text{fixed}}_i \cdot y_{it} + C^{\text{variable}}_{it} \cdot P^{\text{max}}_i \cdot u_{it} \right)
\]

Here, the term $(y_{it} - y_{i,t-1})$ is 1 if power unit i is started at time t and 0 otherwise. Similarly, the term $(y_{i,t-1} - y_{it})$ is 1 if power unit i is shut down at time t and 0 otherwise. The variable u_{it} is the fraction of the maximum output power of power unit i at time t. The variable y_{it} is a binary variable indicating whether power unit i is turned on at time t.

User
We are now at the third step.

Mixtral 8x22B
Step 3: Define the constraints

1. Power output bounds: The power output of each power unit must be within its lower and upper bounds.

\[
P^{\text{min}}_i \cdot y_{it} \leq P_{it} \leq P^{\text{max}}_i \cdot y_{it}, \quad \forall i \in \{1,2,3,4,5,6\}, \forall t \in \{1,2,\ldots,15\}
\]

2. Power balance: The total power output from all power units must meet the total power demand at each time period and be 10% higher than the demand for security reasons.

\[
\sum_{i=1}^{6} P_{it} \geq 1.1 \cdot D_t, \quad \forall t \in \{1,2,\ldots,15\}
\]

3. Power increment and decrement: The change in power output from one time period to the next must not exceed the maximum power increment and decrement for each power unit.

\[
-\Delta^{\text{down}}_i \cdot y_{i,t-1} \leq P_{it} - P_{i,t-1} \leq \Delta^{\text{up}}_i \cdot y_{it}, \quad \forall i \in \{1,2,3,4,5,6\}, \forall t \in \{1,2,\ldots,15\}
\]

4. Binary variables: The binary variables y_{it} must be between 0 and 1.

\[
y_{it} \in \{0,1\}, \quad \forall i \in \{1,2,3,4,5,6\}, \forall t \in \{1,2,\ldots,15\}
\]

5. Continuous variables: The continuous variables u_{it} must be between 0 and 1.

\[
u_{it} \in [0,1], \quad \forall i \in \{1,2,3,4,5,6\}, \forall t \in \{1,2,\ldots,15\}
\]

This completes the formulation of the mathematical optimization model for scheduling the power output of the 6 electric power thermal units over the timespan of 15 periods. The model minimizes the total cost, including startup, shutdown, fixed, and variable costs, while satisfying various operational constraints.

12.1.3.2 Code

User
Please write a Python Pyomo code for this optimization problem. Use sample data where needed. Indicate where you use sample data.

User
Step 1: Define parameters and variables

Let's denote:
- i as the index for power units, where $i \in \{1,2,3,4,5,6\}$
- t as the index for time periods, where $t \in \{1,2,\ldots,15\}$
- C^{startup}_i as the startup cost for power unit i
- C^{shutdown}_i as the shutdown cost for power unit i
- C^{fixed}_i as the fixed cost for power unit i
- C^{variable}_{it} as the variable cost for power unit i at time t
- P^{min}_i as the minimum output power for power unit i
- P^{max}_i as the maximum output power for power unit i
- Δ^{up}_i as the maximum power increment for power unit i

- Δ^{down}_i as the maximum power decrement for power unit i
- D_t as the power demand at time t
- $y_{it} \in \{0,1\}$ as a binary variable indicating whether power unit i is turned on at time t
- $u_{it} \in [0,1]$ as a continuous variable representing the fraction of the maximum output power of power unit i at time t

The power output of power unit i at time t is given by $P_{it} = P^{\text{max}}_i \cdot u_{it}$.
Step 2: Define the objective function

The objective function is the total cost, which includes the startup, shutdown, fixed, and variable costs. We want to minimize this total cost. The objective function can be formulated as follows:

\[
\text{minimize} \sum_{t=1}^{15} \left(\sum_{i=1}^{6} C^{\text{startup}}_i \cdot (y_{it} - y_{i,t-1}) + C^{\text{shutdown}}_i \cdot (y_{i,t-1} - y_{it}) + C^{\text{fixed}}_i \cdot y_{it} + C^{\text{variable}}_{it} \cdot P^{\text{max}}_i \cdot u_{it} \right)
\]

Here, the term $(y_{it} - y_{i,t-1})$ is 1 if power unit i is started at time t and 0 otherwise. Similarly, the term $(y_{i,t-1} - y_{it})$ is 1 if power unit i is shut down at time t and 0 otherwise. The variable u_{it} is the fraction of the maximum output power of power unit i at time t. The variable y_{it} is a binary variable indicating whether power unit i is turned on at time t.
Step 3: Define the constraints

1. Power output bounds: The power output of each power unit must be within its lower and upper bounds.

\[
P^{\text{min}}_i \cdot y_{it} \leq P_{it} \leq P^{\text{max}}_i \cdot y_{it}, \quad \forall i \in \{1,2,3,4,5,6\}, \forall t \in \{1,2,\ldots,15\}
\]

2. Power balance: The total power output from all power units must meet the total power demand at each time period and be 10% higher than the demand for security reasons.

\[
\sum_{i=1}^{6} P_{it} \geq 1.1 \cdot D_t, \quad \forall t \in \{1,2,\ldots,15\}
\]

3. Power increment and decrement: The change in power output from one time period to the next must not exceed the maximum power increment and decrement for each power unit.

\[
-\Delta^{\text{down}}_i \cdot y_{i,t-1} \leq P_{it} - P_{i,t-1} \leq \Delta^{\text{up}}_i \cdot y_{it}, \quad \forall i \in \{1,2,3,4,5,6\}, \forall t \in \{1,2,\ldots,15\}

\]

4. Binary variables: The binary variables $y_{it}$ must be between 0 and 1.

\[
y_{it} \in \{0,1\}, \quad \forall i \in \{1,2,3,4,5,6\}, \forall t \in \{1,2,\ldots,15\}
\]

5. Continuous variables: The continuous variables $u_{it}$ must be between 0 and 1.

\[
u_{it} \in [0,1], \quad \forall i \in \{1,2,3,4,5,6\}, \forall t \in \{1,2,\ldots,15\}
\]

This completes the formulation of the mathematical optimization model for scheduling the power output of the 6 electric power thermal units over the timespan of 15 periods. The model minimizes the total cost, including startup, shutdown, fixed, and variable costs, while satisfying various operational constraints.

Mixtral 8x22B
Here is a Python Pyomo code for the optimization problem described:

```Python
import Pyomo.environ as pyo

# Step 1: Define parameters and variables
model = pyo.AbstractModel()

I = range(1, 7)  # Power units
T = range(1, 16)  # Time periods

model.C_startup = pyo.Param(I, within=pyo.NonNegativeReals)  # Startup cost
model.C_shutdown = pyo.Param(I, within=pyo.NonNegativeReals)  # Shutdown cost
model.C_fixed = pyo.Param(I, within=pyo.NonNegativeReals)  # Fixed cost
model.C_variable = pyo.Param(I, T, within=pyo.NonNegativeReals)  # Variable cost
model.P_min = pyo.Param(I, within=pyo.NonNegativeReals)  # Minimum output power
model.P_max = pyo.Param(I, within=pyo.NonNegativeReals)  # Maximum output power
model.Delta_up = pyo.Param(I, within=pyo.NonNegativeReals)  # Maximum power increment
model.Delta_down = pyo.Param(I, within=pyo.NonNegativeReals)  # Maximum power decrement
model.D = pyo.Param(T, within=pyo.NonNegativeReals)  # Power demand

model.y = pyo.Var(I, T, within=pyo.Binary)  # Binary variable indicating whether power unit i is turned on at time t
model.u = pyo.Var(I, T, within=pyo.NonNegativeReals, bounds=(0, 1))  # Continuous variable representing the fraction of the maximum output power of power unit i at time t
model.P = pyo.Var(I, T, within=pyo.NonNegativeReals)  # Power output of power unit i at time t

# Step 2: Define the objective function
def total_cost(model):
    return sum(model.C_startup[i] * (model.y[i, t] - model.y[i, t-1]) + model.C_shutdown[i] * (model.y[i, t-1] - model.y[i, t]) + model.C_fixed[i] * model.y[i, t] + model.C_variable[i, t] * model.P_max[i] * model.u[i, t] for i in I for t in T)

model.OBJ = pyo.Objective(rule=total_cost, sense=pyo.minimize)

# Step 3: Define the constraints
def power_output_bounds(model, i, t):
    return model.P_min[i] * model.y[i, t] <= model.P[i, t] <= model.P_max[i] * model.y[i, t]

model.power_output_bounds_constraint = pyo.Constraint(I, T, rule=power_output_bounds)

def power_balance(model, t):
    return sum(model.P[i, t] for i in I) >= 1.1 * model.D[t]

model.power_balance_constraint = pyo.Constraint(T, rule=power_balance)

def power_increment_decrement(model, i, t):
    if t == 1:
        return pyo.Constraint.Skip
    else:
        return -model.Delta_down[i] * model.y[i, t-1] <= model.P[i, t] - model.P[i, t-1] <= model.Delta_up[i] * model.y[i, t]

model.power_increment_decrement_constraint = pyo.Constraint(I, T, rule=power_increment_decrement)

# Step 4: Solve the optimization problem
# To solve the optimization problem, we need to provide the values for the parameters (C_startup, C_shutdown, C_fixed, C_variable, P_min, P_max, Delta_up, Delta_down, D).
# Once the parameters are provided, we can create a concrete model and solve it using a solver.
```

In this code, the parameters (`C_startup`, `C_shutdown`, `C_fixed`, `C_variable`, `P_min`, `P_max`, `Delta_up`, `Delta_down`, `D`) are defined as abstract parameters, which means that their values will be provided when a concrete model is created. You can provide the values for these parameters based on your specific scenario.