# OptiChat: Bridging Optimization Models and Practitioners with Large Language Models

Hao Chen[1], Gonzalo Esteban Constante-Flores[1], Krishna Sri Ipsit Mantri[1], Sai Madhukiran Kompalli[1], Akshdeep Singh Ahluwalia[1], Can Li[1*]

[1*]Davidson School of Chemical Engineering, Purdue University, West Lafayette, 47907, IN, USA.

*Corresponding author(s). E-mail(s): canli@purdue.edu;
Contributing authors: chen4433@purdue.edu; geconsta@purdue.edu; mantrik@purdue.edu; skompall@purdue.edu; ahluwala@purdue.edu;

**Abstract**

Optimization models have been applied to solve a wide variety of decision-making problems. These models are usually developed by optimization experts but are used by practitioners without optimization expertise in various application domains. As a result, practitioners often struggle to interact with and draw useful conclusions from optimization models independently. To fill this gap, we introduce OptiChat, a natural language dialogue system designed to help practitioners interpret model formulation, diagnose infeasibility, analyze sensitivity, retrieve information, evaluate modifications, and provide counterfactual explanations. By augmenting large language models (LLMs) with functional calls and code generation tailored for optimization models, we enable seamless interaction and minimize the risk of hallucinations in OptiChat. We develop a new dataset to evaluate OptiChat's performance in explaining optimization models. Experiments demonstrate that OptiChat effectively delivers autonomous, accurate, and instant responses. These findings highlight the potential of LLMs to bridge the gap between optimization models and practitioners in the real-world decision-making process.

# 1 Introduction

Significant progress in large language models (LLMs) has been witnessed in recent years [1, 2], with applications emerging in chemistry [3–6], biology [7, 8], healthcare [9, 10], finance [11–13], manufacturing [14, 15], supply chain management [16], and optimization modeling [17–22]. This widespread applicability highlights the capability of LLMs to comprehend and process natural language across diverse domains. Leveraging the versatility of LLMs, a recent study introduced an interactive dialogue system, *TalktoModel* [23], designed to assist practitioners in understanding complex machine learning models from various application domains by integrating LLMs with explainable AI (XAI) techniques [24–27]. Case studies demonstrated that the interactive dialogue system of *TalktoModel* significantly enhanced healthcare workers' understanding of an ML-based disease prediction model.

In addition to machine learning models, optimization models are another type of complex model widely used across various fields, such as engineering, economics, healthcare, and manufacturing [28]. These models formulate real-world decision-making problems into optimizing an objective under a set of constraints (e.g., budgets and operational limits), where each decision variable has a well-defined physical meaning. For example, a supply chain optimization model aims at determining the most economically efficient production levels and product distribution while meeting customer demands. While optimization models are built on structured and interpretable mathematical formulations that are accessible to experts, practitioners face significant challenges in understanding the abstract formulations and often perceive them as black boxes. This challenge becomes even more pronounced when the solutions suggested by these models deviate from familiar heuristics, leaving practitioners uncertain whether to trust them. To address these challenges, the optimization community has also developed explanation techniques analogous to those in XAI. Since the 1980s, the community has developed expert systems [29, 30], interpretable decision rules [31–33], argumentation-based methods [34–36], and counterfactual explanations [37].

However, the effective use of these explanation techniques still requires substantial optimization expertise. It can still be challenging for the end users of the optimization models, such as logistics coordinators, to independently comprehend the model's outcomes, reconcile results that conflict with their own experiences, or conduct further analysis. This limitation imposes a significant burden on the optimization experts tasked with communicating the results to practitioners, slows decision-making processes, and prohibits the dissemination of optimization models in areas where optimization experts are inaccessible.

To address this limitation, a natural approach is to leverage LLMs for explaining optimization models, given their versatility in natural language-based tasks. However, the hallucination of existing LLMs is concerning because it can produce unfounded or inaccurate explanations [38]. One solution proposed by [16] is to rely entirely on code generation to handle all queries for a supply chain optimization model. Alternatively, our previous work [39] augments LLMs with predefined functions, but this system is specifically designed for diagnosing infeasible optimization models. Moreover, there is currently no general-purpose platform tailored for explaining optimization models

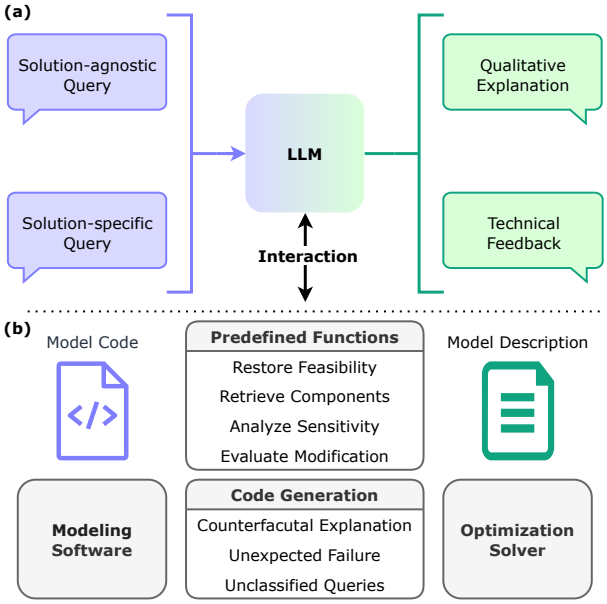


**Fig. 1** Overview of OptiChat. Inputs in blue, outputs in green, and tools in grey. (a) User interface. During a conversation, the solution-agnostic query is answered directly through the LLM's in-context reasoning. The solution-specific query is addressed by interacting with several optimization tools and the model. (b) Backend system. The model code is preprocessed into a natural language description, both of which are accessible to the LLM and the optimization tools. The optimization tools include an optimization solver, an algebraic modeling language, predefined functions, and code generation.

using natural language, nor are there open-source datasets available for systematically evaluating such systems.

In view of these challenges, we introduce OptiChat, an LLM-assisted system designed to not only interpret optimization models in natural language but also provide post hoc explanations through interactive dialogues after the optimization model is solved. Common types of user queries can be classified and systematically addressed by OptiChat. To achieve this, we combine various predefined functions and code generation to draw faithful and trustworthy conclusions from optimization models. We also curate a comprehensive dataset to evaluate OptiChat, supporting further research in this area.

As illustrated in Figure 1, the input to OptiChat is a well-written code script corresponding to an optimization model in the Pyomo/Python [40] algebraic modeling language, developed by an optimization expert. OptiChat initially processes the code to generate a coherent and easy-to-understand description to assist practitioners in understanding the problem context. After that, practitioners can ask queries of interest in an interactive dialogue. We survey the queries gathered from practitioners and

**Diagnosing Query**: I'm having trouble obtaining a solution from the model. By how much do I need to adjust the maximum production capacities to fix this?

**Diagnosing Action**: Add `model.slack_pc["max", :]` and solve a problem to minimize `sum(model.slack_pc["max", i] for i in model.units)`
**Diagnosing Answer**: You must increase the maximum production capacities at the 1st facility by at least 50 units if you want to make the model feasible.

**Retrieval Query**: What do you believe are the best aircraft assignments for the ORD-SAN route?

**Retrieval Action**: Retrieve the values of binary variables `model.x[:, "ORD-SAN"]`
**Retrieval Answer**: The model suggests assigning aircraft #4 and #7 to this route.

**Sensitivity Query**: Winter is coming. How will our total profit be affected by the seasonal fluctuation in customer orders?

**Sensitivity Action**: Find the values of `model.dual[model.orders_constr]`
**Sensitivity Answer**: The profit turns out to be sensitive to fluctuations in customer orders, with any unit decrease leading to an $86.3 loss

**What-if Query**: Can our plant still meet demand if the national regulation now cuts the limit of carbon dioxide emissions by 10%?

**What-if Action**: Modify `model.emission["CO2"].value *= 0.9` and resolve the problem
**What-if Answer**: You can still meet all the demands despite the stricter CO2 emission limit, though the total profit has been lowered to $ 297,874.

**Why-not Query**: Why is it not recommended to at least build a steam boiler or a furnace to supply sufficient heat?

**Why-not Action**: Add constraint `model.y["boiler"] + model.y["furnace"] >= 1` and resolve the problem
**Why-not Answer**: If so, building a steam boiler will push the optimal cost up to $1.329M, which is much higher than the previous $0.673M

**Fig. 2** Illustrative examples of five different types of queries, the proposed actions, and answers. The proposed actions represent the strategies that generate the relevant explanations, implemented in OptiChat through predefined functions or code generation. The answers are concise summaries of the responses produced by OptiChat.

classify them into five categories: diagnosing, retrieval, sensitivity, what-if, and why-not queries (see illustrative examples in Figure 2). Diagnosing queries focus on fixing infeasibility issues. Retrieval queries seek to extract relevant model data. Sensitivity queries measure the impact of parameter uncertainties on the optimal objective value. What-if queries evaluate significant changes in input parameters specified by users. Lastly, why-not queries investigate the counterfactual scenarios suggested by users. These are termed as *solution-specific queries*, meaning OptiChat must derive a solution from the model to provide an accurate explanation. Underneath, explanation strategies for each query category are either implemented as predefined functions or generated as code by the LLM, which are then executed using an optimization solver, such as Gurobi. These two approaches are tailored to achieve high correctness rates in answering queries. Besides the *solution-specific queries*, the users can also seek qualitative explanations from the LLM. These queries, termed as *solution-agnostic queries*,

are contextualized within the LLM's memory using the input optimization model and the chat history to ensure the quality of the answers.

## 2 Related Works

We give a brief review of recent applications of large language models in operations research. We divide the works into three different categories which correspond to the typical workflow of developing and adopting an optimization model. (1) Use LLM to formulate optimization models based on the problem statement. (2) Use LLM to help develop a new optimization algorithm (3) Use LLM to facilitate interaction with with the user. Our work belongs to the third category.

**Formulating optimization models** OptiMUS [17, 19] is a multi-agent LLM-based system designed to formulate and solve (mixed-integer) linear programming problems from natural language descriptions. The system can develop mathematical models, write and debug solver code, evaluate generated solutions, and iteratively improve model and code efficiency and correctness based on these evaluations. [41] proposed ORLM (Operations Research Language Model), a novel approach to OR modeling that relies on fine-tuning a semi-synthetic dataset rather than using prompt engineering or agentic models. [20] introduced a multi-agent cooperative framework called Chain of Experts (CoE) for automated OR problem modeling. [42] developed a Monte Carlo Tree Search (MCTS)-based approach that decomposes the modeling process into sequential stages (e.g., variables, objectives, constraints) and uses MCTS to explore plausible model structures. [43] presented OptiBench, a benchmark suite for end-to-end optimization problem solving with human-readable inputs and outputs.

**Develop novel optimization algorithms.** Another line of work focuses on the development of novel OR algorithms. *FunSearch* [44], which explores the function space, is an evolutionary procedure that pairs a pretrained large language model (LLM) with a systematic evaluator. It has been successfully applied to enhance online bin packing heuristics by evolving the heuristics generated by the LLM. A genetic programming algorithm is employed to balance exploitation and exploration within the database of LLM-generated programs. Building on the pioneering efforts of *FunSearch*, several subsequent works have extended similar methodologies to other combinatorial optimization problems [45].

**Facilitating user interactions with optimization models** The final research direction focuses on using LLMs to ease user interactions with optimization models. [46] demonstrated the use of LLMs to allow users to customize meeting preferences in a constraint programming-based scheduling model. [47] designed an LLM-based system for travel planning. [48] applied LLMs to explain solutions to vehicle routing problems. Closest to our framework is OptiGuide [16], which is a multi-agent system for explaining supply chain models by relying entirely on LLM-based code generation to answer user queries. Our main innovation lies in using predefined functions to improve the accuracy and the speed of the agentic framework, as demonstrated through our ablation studies. Compared with the agentic framework of [16], OptiChat has syntax reminders and Operator agents to coordinate the execution of the predefined functions. We also develop a dataset to evaluate our proposed framework on applications across

multiple domains, rather than being restricted to a single use case. This work builds upon our earlier system [39] for diagnosing infeasibility, which was not agentic and did not support queries related to optimality.

# 3 Methods

OptiChat is composed of a user interface, a structured sequence of LLM-based agents, an optimization solver, a modeling software, and several custom-built functionalities. In this work, OptiChat utilizes GPT-4 [1] for the LLM-based agents, Gurobi [49] as the optimization solver, and Pyomo [40] as the algebraic modeling language. In this section, we first motivate the functionalities of OptiChat by providing background on common problems faced by practitioners and connecting them with the proposed explanation strategies. We then present the design of our multi-agent framework, outlining the role and sub-task of each agent. Lastly, the exception management for specific queries is discussed. The implementation of OptiChat is avaiable on GitHub: https://github.com/li-group/OptiChat.git.

## 3.1 Problem Statement And Explanation Strategies

**3.1.1 Feasible / Infeasible Model Description** Most practical applications in optimization can be formulated as (mixed-integer) linear programs (MILP/LP):

$$\begin{aligned} \min_{\mathbf{x}} \quad & \mathbf{c}^\top \mathbf{x} \\ \text{subject to} \quad & \mathbf{A}\mathbf{x} \leq \mathbf{b}, \\ & \mathbf{x} \in \mathbb{Z}^p \times \mathbb{R}^{n-p}. \end{aligned} \tag{1}$$

where decisions to make are denoted as $\mathbf{x}$ that can consist of both integer and continuous variables; $\mathbf{c}$ represents the cost coefficients; the problem is subject to linear constraints $\mathbf{A}\mathbf{x} \leq \mathbf{b}$: $\mathbf{A} \in \mathbb{R}^{m \times n}$, $\mathbf{b} \in \mathbb{R}^m$. However, practitioners often struggle to understand the mathematical formulation. More importantly, optimization models derived from real-world problems can be infeasible at the initial stage. Infeasibility does not refer to code bugs that prohibit models from being executed, but rather to an optimization model in which no solution exists to satisfy all constraints simultaneously. This usually occurs because of overly restrictive constraints and inaccuracies in the underlying data. For example, in an aggressive overselling scenario, an aircraft may lack sufficient seats to accommodate all ticketed passengers. This infeasibility further complicates practitioners' ability to identify flaws in the model.

An irreducible infeasible subset, IIS, is a minimal set of constraints and/or variable bounds within an optimization model that causes infeasibility, defined by two key properties: (i) the IIS itself is infeasible, and (ii) any proper subset of the IIS is feasible. In other words, we can use the IIS to extract the components in violation and characterize the nature of infeasibility. Different algorithms have been developed to isolate IIS, such as deletion filter [50], additive method [51], and hybrid approach [52]. Commercial optimization solvers like CPLEX [53] and Gurobi [49] have implemented

variants of these IIS detection algorithms. For a comprehensive review of IIS detection, readers are referred to the monograph [54].

**3.1.2 Diagnosing Query** Restoring feasibility can be approached in various ways, but not all are actionable in the real world. To restore feasibility, practitioners seek strategies that align with their operational priorities. For instance, suppose a model is infeasible to produce a chemical at a 99% concentration with the current resources available in a chemical plant. Increasing the conversion rate of an existing chemical reaction is generally impractical, while negotiating with business partners to lower product purity requirements is more actionable.

To address this need, one can either (1) recursively remove constraints from the IIS until the model becomes feasible, or (2) add slack variables to the optimization problem and allow for adjustments to input parameters. The second approach is more practical as each constraint reflects an important aspect of the problem and cannot always be removed in practice. In contrast, introducing slack variables into the optimization problem in (1) provides practitioners with a more concrete and actionable plan for feasibility restoration. Mathematically, the following extended problem is solved

$$\begin{aligned} \min_{\mathbf{x},\delta\mathbf{A}^+,\delta\mathbf{A}^-,\delta\mathbf{b}^+} \quad & \sum_{(i,j)\in\mathcal{S}_A} \left(\delta\mathbf{A}_{ij}^+ + \delta\mathbf{A}_{ij}^-\right) + \sum_{i\in\mathcal{S}_b} \delta\mathbf{b}_i^+ \\ \text{subject to } & (\mathbf{A} + \delta\mathbf{A}^+ - \delta\mathbf{A}^-)\mathbf{x} \leq \mathbf{b} + \delta\mathbf{b}^+, \\ & \mathbf{x} \in \mathbb{Z}^p \times \mathbb{R}^{n-p}, \quad \delta\mathbf{A}^+, \delta\mathbf{A}^-, \delta\mathbf{b}^+ \geq 0, \end{aligned} \tag{2}$$

where $\delta\mathbf{A}^+$, $\delta\mathbf{A}^-$, $\delta\mathbf{b}^+$ are nonnegative slack variables. Since all the constraints are inequalities, adding nonnegative slacks $\delta\mathbf{b}^+$ on the right-hand-side relaxes the problem. In contrast, adding the left-hand-side slacks, $\delta\mathbf{A}^+$ and $\delta\mathbf{A}^-$, are less desirable. It is worth noting that because only a subset of parameters can be adjusted in practice, the slack variables differ in dimensionality from the parameters $\mathbf{A}$ and $\mathbf{b}$. The objective is to minimize the total perturbation to the original problem by summing up all the slack variables. In principle, different weights could be assigned to different slack variables, representing the cost of perturbing the corresponding parameters. The optimal solution found in (2) can be explained as the minimal adjustments needed to restore feasibility.

It should be noted that adding slack variables to the left-hand-side parameters $\mathbf{A}$ will lead to a product of variables between $\delta\mathbf{A}^+$, $\delta\mathbf{A}^-$ and $\mathbf{x}$ in (2). This results in a non-convex mixed-integer quadratically constrained program (MIQCP), which is often prohibitive to solve. In many situations, left-hand-side parameters represent immutable properties. For example, in the constraint $\mathbf{Ax} \leq \mathbf{b}$ where $\mathbf{b}$ is a deadline, $\mathbf{x}$ indicates task status, and $\mathbf{A}$ represents processing times. The processing times are an inherent property of the machines and cannot be changed. When OptiChat detects a request to add slacks to $\mathbf{A}$, it will alert users of this immutability before initiating the MIQCP.

**3.1.3 Retrieval Query** An optimization model consists of decision variables, parameters data, constraints, and an objective function. Optimization models developed for practical use are often large in scale, with some components indexed over large sets. For example, a variable that represents the assignment decision of an aircraft

can be indexed over hundreds of routes. When practitioners need to review specific data or optimal decisions from the model, it is inefficient to retrieve this information. OptiChat facilitates real-time access to specific model information through natural language.

**3.1.4 Sensitivity Query** Many optimization models are constructed with incomplete knowledge of problem parameters. For example, electricity prices and customer demands often fluctuate in the market over time and cannot be fixed within the model. The values of these parameters can be predicted using historical data and updated on a rolling basis. It is important for practitioners to evaluate how changes in problem parameters impact the optimal objective value, undertake risk assessment, and devise appropriate management strategies.

We propose to perform sensitivity analysis based on well-established duality theory in linear programming (LP) [55] where $\mathbf{x}$ does not contain integer variables. In short, the change in the optimal objective value in response to changes in input parameters can be expressed as a value function

$$v(\mathbf{b}) = \min_{\mathbf{x}\in\mathcal{X}} \mathbf{c}^\top \mathbf{x} = {\mathbf{y}^*}^\top \mathbf{b}, \tag{3}$$

where $\mathcal{X} = \{\mathbf{x} : \mathbf{A}\mathbf{x} \leq \mathbf{b}\}$ and ${\mathbf{y}^*}^\top$ is the optimal solution to the dual problem of (1). Problem (1) called a primal problem that finds the minimum cost while satisfying all the constraints, whereas the dual problem of (1) aims to seek the tightest lower bound on the optimal primal cost. Denote the original data given in the primal problem as $\mathbf{b}^*$, the expression (3) indicates that when the perturbed parameters $\mathbf{b}$ are close to $\mathbf{b}^*$, the optimal primal objective value is a linear function of the input parameters with gradient $\mathbf{y}^*$. In other words, it precisely reflects the sensitivity of the optimal objective value in the vicinity of $\mathbf{b}^*$. The access to dual information in LP is supported by most optimization solvers [49, 53, 56].

**3.1.5 What-if Query** Although sensitivity analysis can evaluate local changes in parameters, it is not a valid methodology to evaluate larger changes. This limitation arises because the optimal dual solution in (3) is not constant but depends on parameters. After large parameter perturbations, the dual solution previously found in the original problem no longer accurately reflects the change in optimal objective value. These significant perturbations often occur when a new policy is being established, or the industry is evaluating a new business strategy. For example, practitioners may pose questions such as "What if we increase the labor force to 35 people?" and "What if customer orders are cut by a third?"

In these cases, OptiChat systematically identifies the extent and type of modification from the user's query. The model (1) is then revised by updating parameters and constraints accordingly:

$$\begin{aligned} \min_{\mathbf{x}} \quad & \hat{\mathbf{c}}^\top \mathbf{x} \\ \text{subject to} \quad & \hat{\mathbf{A}}\mathbf{x} \leq \hat{\mathbf{b}}, \\ & \mathbf{x} \in \mathbb{Z}^p \times \mathbb{R}^{n-p}. \end{aligned} \tag{4}$$

In this revision, $\hat{\mathbf{c}}$, $\hat{\mathbf{A}}$ and $\hat{\mathbf{b}}$ are completed by the interaction between LLMs and our predefined function. Modifications will change the feasible region of the problem, generally leading to a different optimal solution.

**3.1.6 Why-not Query** Mathematical optimization is a rigorous methodology that searches for the optimal solution. In contrast, practitioners often rely on their experience to make decisions. As a result, it can be challenging to convince business managers or stakeholders to accept the optimal solutions given by an optimization model, especially when these solutions are against their intuitions. For example, one might ask: "Why not choose supplier 1?" in a supply chain model, even though the current optimal solution does not include it.

A counterfactual explanation addresses the "why-not" queries by examining alternative decisions or outcomes that are not currently realized in the optimal solution. To investigate such a counterfactual scenario, we modify the original model by incorporating an additional set of constraints that force the desired alternative to occur. In this example, if $x_1$ represents the binary decision to select supplier 1, we add the constraint $x_1 = 1$ to enforce this choice. Once we re-solve the modified problem with this counterfactual constraint in place, we can observe how the objective value and the feasibility of the solution change. For instance, the forced selection of supplier 1 may raise production costs, reduce overall profitability, or even make the problem infeasible. By comparing the new solution to the original one, we gain insight into the trade-offs and underlying reasons that the original optimal solution did not include supplier 1. This counterfactual reasoning thus provides a transparent and actionable explanation of the model's decision-making process, illustrating which model parameters and constraints are most influential in ruling out the queried alternative. The explanation for the "why-not" query relies on code generated by the LLM since the counterfactual can be any arbitrary fact suggested by the practitioners, for which predefined functions do not suffice.

## 3.2 Multi-agent Framework

Figure 3 depicts the workflow of the multi-agent framework. Before the interactive dialogue, OptiChat begins with the Illustrator agent, which preprocesses the Pyomo optimization model uploaded by users. During the conversation, the Coordinator routes queries either to the Explainer agent, when they can be directly answered by an LLM, or to the team of Engineering agents when more specialized reasoning is required. The Engineering team operates in a structured order, comprising the following agents: (1) the Operator, who controls all predefined functions; (2) the Programmer and Evaluator agents, responsible for generating code for tasks that cannot be addressed by the predefined functions; and (3) the Reminder agent, who provides supplementary information to enhance the accuracy of the other two. The prompts are shown in Appendix C. To illustrate the workflow, a what-if query is provided as an illustrative example in Appendix D.

**3.2.1 Illustrator** This agent extracts sets, parameters, variables, constraints, and objectives from the Pyomo optimization model, labeling each component with a natural language description. These descriptions establish a dictionary that maps the physical meanings of model components to their corresponding notations in the source

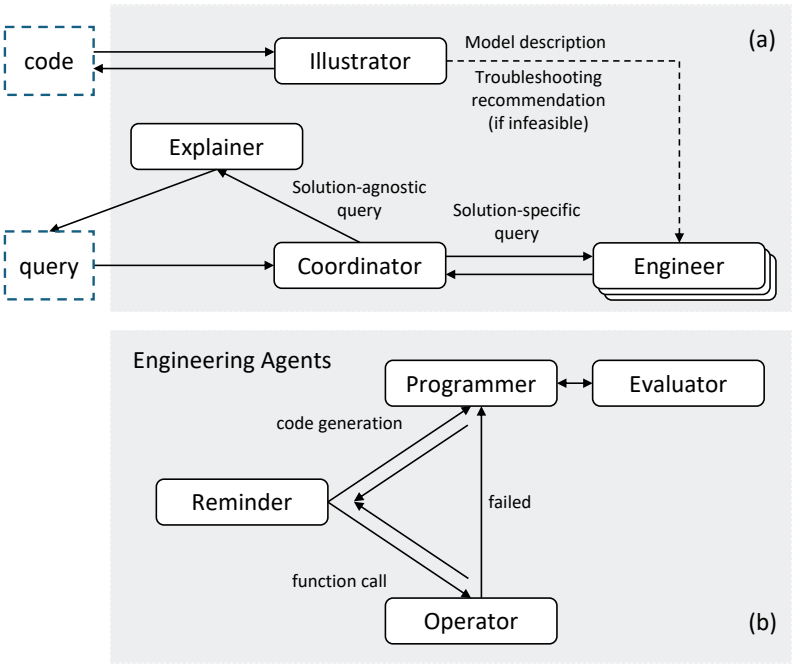


**Fig. 3** Multi-agent framework of OptiChat. (a) The Pyomo model code is interpreted by the Illustrator agent. Solution-agnostic queries are addressed by the Explainer agent directly. Solution-specific queries are handled by a team of Engineering agents, followed by the Explainer agent. (b) The Engineering agents include the Reminder, Operator, Programmer and Evaluator agents.

code. Since users may refer to model components using different terms, this preprocessing enables LLMs to accurately identify the referenced component by providing the problem context. Alongside these descriptions, information about components, such as index dimensions and solution status, is also stored to support interaction with optimization tools. Finally, the Illustrator agent describes the model to users in a concise and coherent manner using natural language. If the solution status of a model indicates infeasibility, the agent will also invoke tools to compute the IIS and provide troubleshooting recommendations by interpreting the conflicting constraints.

**3.2.2 Coordinator** After the Illustrator agent preprocesses a model, users can submit queries to initiate an interactive conversation. The Coordinator agent classifies each query as solution-agnostic or solution-specific. Solution-agnostic queries are solely addressed by the Explainer agent through in-context reasoning, while solution-specific queries require technical feedback from agents in the Engineering team before being forwarded to the Explainer agent.

**3.2.3 Explainer** The Explainer agent is responsible for conveying any technical information to practitioners in an understandable manner as the endpoint for all queries.

**3.2.4 Reminder** The Reminder agent serves as the entry point for every solution-specific query, guiding LLMs to more accurately determine which function to invoke, which model component to modify, and which specific index within that component to

reference. Among these tasks, selecting the correct component index (e.g., a parameter index) is the most error-prone. In contrast, identifying the appropriate function name is generally easier for LLMs, aided by few-shot demonstrations. The model description provided by the Illustrator agent helps the Reminder to pinpoint the relevant component name. The syntax guidance offered by the Reminder agent has been shown in our ablation studies (Section 4.2.4) to improve the accuracy in identifying the correct function, component names and indices.

**3.2.5 Operator** As proposed in Section 3.1, we implement four predefined functions in the Operator agent to address the diagnosing, retrieval, sensitivity, and what-if queries in Figure 2. Informed by the specific syntax guidance, this agent selects the appropriate function and arguments, then invokes the corresponding tool to generate the solution.

**3.2.6 Programmer And Evaluator** The development of the Programmer and Evaluator agents is inspired by recent works on applying code generation to optimization [16, 18]. The code generation capability of LLMs has been widely explored and demonstrated impressive results across various tasks [57]. These agents are designed to address the why-not queries in Figure 2, but also potentially handle unexpected queries beyond the reach of predefined functions. The Programmer and Evaluator are involved in a loop with limited iterations. The Evaluator first executes the code generated by the Programmer, then reviews the terminal outputs and error messages if available. Throughout this process, the code is automatically refined until a bug-free and comprehensive solution is produced. The prompts for why-not queries are designed to guide the LLM in generating additional constraints from the user's counterfactual query, thereby narrowing the LLM's task scope and reducing its code output.

## 3.3 Exceptions Management

When practitioners interact with OptiChat, the system is robust in handling exceptions caused by the lack of optimization expertise. The sensitivity analysis in OptiChat relies on the strong duality of LP. However, practitioners may request sensitivity analysis concerning left-hand-side parameters $\mathbf{A}$ in LP models or parameters in MILP models. Unlike the sensitivity of $\mathbf{b}$ in (3), the dependence of the optimal objective value on left-hand-side parameters $\mathbf{A}$ cannot be determined based on duality theory. When the optimization model is an MILP, the equality in (3) no longer holds, breaking the connection between the optimal objective value and $\mathbf{b}$. In this case, OptiChat will notify them that sensitivity analysis is not supported and suggest providing specific modifications for evaluation if they still wish to address the same queries. This converts a sensitivity query into a what-if query, which can be addressed in a less restrictive manner. Similarly, when OptiChat detects a request to add slacks to $\mathbf{A}$ in (2), it will issue a warning message to indicate the potential immutability of left-hand-side parameters $\mathbf{A}$ and the increased processing time required to initiate the MIQCP. The model information, such as whether a parameter appears on the left-hand-side, is stored during the preprocessing step. This information will be automatically used to verify the presence of such exceptions in the predefined functions and guide the user accordingly. Furthermore, when unexpected failures occur in predefined functions,

the Programmer and the Evaluator agents will be invoked to generate code-based solutions.

# 4 Results and Discussion

In this section, the effectiveness of OptiChat is demonstrated in terms of model descriptions and query responses. First, we emphasize the efficiency of OptiChat in generating autonomous model descriptions compared to consulting with optimization experts. In terms of the interactive dialogue, we measure the correctness rates of responses for each query type, providing a quantitative evaluation of OptiChat's accuracy. We also present a qualitative showcase of query responses, illustrating how OptiChat assists practitioners in real-world settings.

We test OptiChat on 24 optimization models written in the Pyomo/Python framework. To demonstrate the versatility of LLMs, the 24 models span various contexts, including supply chain, manufacturing and production, petroleum refinery, industrial scheduling, chemical and process system engineering, transportation, etc. The 24 models are adapted from the GAMS Model Library[1], sourced from the Pyomo Cookbook by the University of Notre Dame[2] and a public GitHub tutorial[3], or adapted from an optimization textbook [28]. Infeasible variants of these models are created by adjusting the model parameters or introducing additional constraints. A summary of the statistics including the number of variables, constraints, and parameters is shown in Appendix A. The size of the models are orders of magnitude larger than those in existing natural language to OR models benchmarks.

OptiChat aims to make optimization models more accessible to a wider audience by enabling seamless interaction between users and the underlying models, thereby reducing the time experts must spend communicating with users. For our experiments, we recruited 29 experts, including graduate students and postdoctoral researchers, each with at least one year of experience in optimization theory or modeling. These experts were tasked with drafting model descriptions and answering follow-up queries.

## 4.1 Model Description

In this study, the expert participants were required to write detailed model descriptions based on the Pyomo scripts of the optimization models. If a model was found to be infeasible, the expert participants were also responsible for diagnosing the source of the infeasibility. Following these tasks, the experts were surveyed to provide time estimates for completing them. Given the differences in expertise among the participants and the varying complexity of the assigned models, we report the most frequently selected time range in the survey.

On the other hand, OptiChat automatically interprets optimization models in natural language without involving optimization experts, which significantly shortens the time. More importantly, both the experts and OptiChat are augmented with tools to isolate the irreducible infeasible subset (IIS, see details in Section 3.1) for

[1] https://www.gams.com/latest/gamslib_ml/libhtml/index.html
[2] https://jckantor.github.io/ND-Pyomo-Cookbook
[3] https://github.com/hdavid16/RTN-Demo

characterizing infeasible optimization models. By isolating the conflicting constraints, optimization experts can gain valuable insight from the IIS analysis. However, it is still time-consuming for them to identify root causes and take corrective actions. In contrast, OptiChat automates the troubleshooting process along with a model description within one minute, as shown in Table 1. Empirically, the quality of OptiChat's description is observed to be comparable to those provided by experts, which can be attributed to the versatility of LLMs in different problem contexts. Details of the expert survey used to evaluate the model descriptions generated by the LLM are provided in Appendix B.

**Table 1** Model Description Results.

| | OptiChat | Expert |
|---|---|---|
| Model | Time (min) | Time (min) |
| Feasible | $< 1$ | $18 - 40$ |
| Infeasible | $< 1$ | $23 - 55$ |

In practical applications, the model description in natural language is often already available before the model code is developed, so having the LLM generate such descriptions is not always necessary. Nevertheless, this study serves two important purposes. First, it demonstrates the flexibility of the LLM in explaining why a model is infeasible when an infeasible instance is encountered after model development. Second, by comparing the LLM-generated descriptions with expert answers, we can validate that the outputs produced by the Illustrator agent are sufficiently accurate and reliable to support subsequent query responses.

## 4.2 Query Response Assessment

After reading the autonomous model description, practitioners gain an understanding of the problem context and are prepared to interact with OptiChat by asking queries. Solution-specific queries are addressed in two steps. First, OptiChat either invokes predefined functions or generates code to draw accurate conclusions from the optimization models. Second, OptiChat explains these conclusions in natural language to guarantee their interpretability to practitioners.

**4.2.1 Query Dataset Development** We curate a comprehensive test dataset of 172 question-answer pairs to quantitatively evaluate the accuracy of OptiChat in response to the user's query, with each pair developed in the context of a particular optimization model. The queries and answers were drafted by the 29 experts recruited and each verified by multiple authors of the paper. We classify these questions into 5 categories: (1) diagnosing; (2) retrieval; (3) sensitivity; (4) what-if; (5) why-not queries as exemplified in Figure 2.

The correctness rate of query responses is evaluated using two distinct approaches. For why-not queries that rely on code generation, the dataset provides natural language explanations explicitly containing the optimal objective values of counterfactual models. An LLM is used to compare these ground truth objective values with those

produced by OptiChat. If the new optimal objective value computed by OptiChat matches the value specified in the ground truth answer, the LLM marks the response as correct. In rare cases, however, the LLM may generate incorrect counterfactual constraints while still producing a matching objective value. To ensure reliability, we manually review instances marked correct by the LLM to validate both the generated constraints and the resulting answers. In contrast, diagnosing, retrieval, sensitivity, and what-if queries are addressed through predefined functions. These function calls return results that are directly informative to the query. Across all dataset instances, we observe that the Explainer agent does not hallucinate when interpreting function outputs, provided that the correct function and arguments (e.g., component names and indices) are selected. To enable automatic evaluation, ground truth outputs are structured in JSON format. If both the function name and arguments chosen by OptiChat match the ground truth, the corresponding answers are deemed correct.

**4.2.2 Quantitative Assessment of the Query Responses** Table 2 reports results for four different LLMs: GPT-4o-mini, GPT-4o, GPT-4.1, and o3. The first three are standard pretrained LLMs. o3 is OpenAI's latest reasoning model which features chain-of-thought reasoning at the expense of longer "thinking" time. Among them, o3 achieves the highest average accuracy across all different types of queries. Notably, it achieves significant improved performance on the why-not queries—78.4% compared to 62.2% for GPT-4.1 and 54.1% for GPT-4o—highlighting the value of explicit reasoning capabilities when code generation is required to generate the counterfactual constraints. However, o3 also exhibits higher latency, taking up to 0.9 minutes per response for the "why-not" queries, this trade-off appears. In contrast, the model with smallest size, GPT-4o-mini, underperforms across all categories, suggesting limitations in both scale and reasoning depth. Overall, OptiChat delivers strong accuracy (above 80% in most query types) and fast responses (under one minute), confirming its practicality for interactive use.

**Table 2** The accuracy and average response time of the five types of queries using GPT-4o-mini, GPT-4o, GPT-4.1, and o3. The best-performing accuracy for each type of model is highlighted in bold.

| | Accuracy (%) | | | | Time (min) | | | |
|---|---|---|---|---|---|---|---|---|
| Query | GPT-4o-mini | GPT-4o | GPT-4.1 | o3 | GPT-4o-mini | GPT-4o | GPT-4.1 | o3 |
| Diagnosing | 53.8 | 84.6 | **89.7** | **89.7** | 0.1 | 0.2 | 0.1 | 0.6 |
| Retrieval | 66.7 | 92.3 | 94.9 | **97.4** | 0.1 | 0.1 | 0.1 | 0.3 |
| Sensitivity | 72.2 | **94.4** | **94.4** | **94.4** | 0.1 | 0.2 | 0.1 | 0.5 |
| What-if | 64.1 | **94.9** | 84.6 | 87.2 | 0.1 | 0.2 | 0.1 | 0.4 |
| Why-not | 62.2 | 54.1 | 62.2 | **78.4** | 0.3 | 0.4 | 0.3 | 0.9 |
| Total | 62.8 | 83.1 | 84.3 | **88.9** | 0.2 | 0.2 | 0.2 | 0.5 |

The high accuracy of OptiChat is attributed to two key factors. First, we develop a multi-agent framework to guide LLMs in generating the code for counterfactual constraints. The prompt is specifically tailored for why-not queries rather than general queries, guiding the LLM to generate a minimal amount of code under a narrowed task scope. Second, we incorporate various predefined functions to prevent LLMs from

developing explanatory techniques from scratch for other queries. This approach is more robust than code generation, as demonstrated by the higher correctness rates in the first four queries in Table 2. More importantly, code generation is currently unable to develop more advanced explanatory techniques necessary for diagnosing queries and sensitivity queries. It is observed that code generation only produces heuristic techniques to tackle these queries, which results in suboptimal conclusions. Therefore, predefined functions are indispensable at the current stage. These factors contributing to the high accuracy will be discussed in more detail in the ablation studies (Section 4.2.4).

**4.2.3 Failure Analysis** Despite achieving high accuracy, the LLM may occasionally fail to match a query to the appropriate function or generate code that deviates from the user's intended purpose. To better understand these model failures, we categorize errors into three types: *syntax errors*, *classification errors*, and *logic errors*.

*Syntax errors* include code execution failures and invalid function calls. These issues often arise from the ambiguity of natural language, which may mislead the LLM into generating function arguments or code that are not executable. Practitioners may describe the same model component using different terminology, while only one executable representation is valid in the model. This challenge is further amplified when components are indexed across multiple dimensions. Users may specify only a subset of dimensions, list them in an incorrect order, or omit them entirely. For example, `pc["max", :]` denotes the maximum production capacities for all facilities, yet practitioners might refer to it verbally as *max output limits*, frequently omitting the qualifier *for all facilities*. The LLMs might hallucinate by providing indices that do not exist.

*Classification errors* occur when the explanation strategy selected by the LLM does not align with the human-annotated query type, such as misclassifying a *why-not* query as a *what-if* query.

*Logic errors* arise in two scenarios. First, during code generation, the LLM may fail to represent a counterfactual scenario correctly using constraints. Second, the component names or indices generated by the LLM may exist in the model as valid function arguments, i.e., not a syntax error, but fail to accurately capture the user's intent.

With this categorization in mind, the breakdown of errors using GPT-4.1 and o3 are shown in Table 3. The proportions of syntax and classification errors in o3, 10. 0% and 25. 0%, respectively, are notably lower compared to GPT-4.1, which exhibits 46.8% syntax errors and 31.3% classification errors. This indicates that o3 adheres more reliably to the syntax guidance and few-shot query demonstrations provided in the prompts. As a result, the proportion of logic errors becomes dominant in o3 (65.0%) compared to GPT-4.1 (21.9%).

**4.2.4 Ablation Studies** To evaluate the impact of the proposed multi-agent framework, we perform ablation studies by removing the predefined functions, the syntax reminders, and the Illustrator, respectively. The results are shown in Table 4 for the top two best performing models, GPT-4.1 and o3.

When the predefined functions are removed in Table 4, OptiChat must rely entirely on code generation, i.e., the Programmer agent, to answer all queries. We observe a

**Table 3** Proportion Of Error Types

| Error Type | GPT-4.1 (%) | o3 (%) |
|---|---|---|
| Syntax Error | 46.8 | 10.0 |
| Classification Error | 31.3 | 25.0 |
| Logic Error | 21.9 | 65.0 |

substantial drop in accuracy for both models, especially on the diagnosing and sensitivity queries. This indicates that the LLMs struggle to write code for retrieving the IIS or the dual variables of a constraint, tasks that demand deeper optimization expertise than the retrieval and what-if queries. Although both models suffer accuracy losses, o3 outperforms GPT-4.1, consistent with the claim that o3 is a superior reasoning model that excels at code generation. We also note a slight increase in response time without the predefined functions, reflecting the additional time the LLMs spend generating code versus leveraging existing tools.

**Table 4** Ablation study results including experiments without predefined functions, syntax reminders, and illustrator.

| | | Accuracy (%) | | Time (min) | |
|---|---|---|---|---|---|
| Query | Setting | GPT-4.1 | o3 | GPT-4.1 | o3 |
| Diagnosing | Main | 89.7 | 89.7 | 0.1 | 0.6 |
| | w/o Predefined Functions | 0.0 | 33.3 | 0.2 | 1.5 |
| | w/o Syntax Reminders | 41.0 | 87.2 | 0.1 | 0.5 |
| | w/o Illustrator | 74.4 | 84.6 | 0.1 | 1.0 |
| Retrieval | Main | 94.9 | 97.4 | 0.1 | 0.3 |
| | w/o Predefined Functions | 56.4 | 84.6 | 0.2 | 0.5 |
| | w/o Syntax Reminders | 87.2 | 97.4 | 0.1 | 0.3 |
| | w/o Illustrator | 74.4 | 87.2 | 0.1 | 0.7 |
| Sensitivity | Main | 94.4 | 94.4 | 0.1 | 0.5 |
| | w/o Predefined Functions | 0.0 | 0.0 | 0.3 | 0.9 |
| | w/o Syntax Reminders | 72.2 | 94.4 | 0.1 | 0.4 |
| | w/o Illustrator | 83.3 | 83.3 | 0.1 | 0.8 |
| What-if | Main | 84.6 | 87.2 | 0.1 | 0.4 |
| | w/o Predefined Functions | 82.1 | 92.3 | 0.2 | 0.9 |
| | w/o Syntax Reminders | 66.7 | 84.6 | 0.1 | 0.3 |
| | w/o Illustrator | 76.9 | 84.6 | 0.2 | 0.7 |
| Why-not | Main | 62.2 | 78.4 | 0.3 | 0.9 |
| | w/o Illustrator | 62.2 | 70.3 | 0.4 | 1.9 |

The syntax Reminder provides guidance for retrieving the correct arguments for predefined functions. When the Reminder is disabled, as shown in Table 4, we observe a slight accuracy drop in the o3 model and a significant drop in GPT-4.1. This further confirms the stronger reasoning capabilities of o3, while the weaker model tends to hallucinate without the syntax Reminder. The difference in response time is marginal, with or without the syntax Reminder.

**User**: How will total profit change if the maximum production capacity changes?

**OptiChat**: ... **Facility 1:** ...
**Facility 2:** Adjustments to maximum production capacity significantly affect the total profit with an increase by 40.20 units of profit. Increasing the overtime production capacity at this facility also helps achieve a better profit margin, adding an additional 27.20 units of profit.
**Facility 3:** Like Facility 2, an increase in the maximum production capacity improves total profit by 40.20 units of profit. **A profound impact is observed when increasing the overtime production capacity at Facility 3, with a rise in profit by an impressive 191.20 units**...

**Fig. 4** A conversation with OptiChat. a, OptiChat helps practitioners identify sensitive parameters.

**User**: Why don't we just go with one or less recovery center? I thought this was a better plan since the investment is so high.

**OptiChat**: ... attempting to limit the logistics model to only one recovery center results in an infeasible solution. This means **it is not possible to meet all operational requirements with just one recovery center**...Two recovery centers are necessary to handle the volume of recoverable products efficiently. A single center might not have sufficient capacity, leading to operational bottlenecks...

**Fig. 5** A conversation with OptiChat. b, OptiChat helps practitioners embrace unanticipated decisions.

The Illustrator agent extracts model components and their descriptions into a lookup table, providing essential context for answering various queries. This context is crucial for OptiChat's performance. As shown in Table 4, both models experience a notable drop in accuracy without the Illustrator. Moreover, the response time increases significantly because the models must reprocess the original code to answer follow-up queries, rather than leveraging the pre-extracted context.

**4.2.5 Qualitative Assessment of Query Responses** We showcase some insightful answers produced by OptiChat. The models used for the demonstration are built for supply chain management, which decides the optimal production, storage, and transportation of goods across production facilities, distribution centers, and markets. The maximum capacities of the normal and the overtime production often fluctuate due to factors such as raw material availability and labor engagement. Consider a case where a manufacturer is informed about an increase in raw materials and plans to expand maximum production capacities at certain facilities. As suggested in Figure 4, the optimal profit is highly sensitive to the maximum overtime production capacity at the third facility, even though overtime production is commonly perceived as costly. Consequently, during the strategy development phase, it is crucial to prioritize the third facility over others and produce these additional items at the overtime stage.

The second model takes into account recovery centers that manage remanufacturing and recovery activities. This model suggests building two such recovery centers, despite the substantial investment cost typically associated with each. Suppose a situation where business managers express concerns regarding the financial burden and are inclined to construct fewer recovery centers. To reassure them, OptiChat justifies this decision by demonstrating that no feasible solution exists if forcing the number of recovery centers no greater than one in Figure 5. Therefore, business managers can be convinced to deprecate outdated practices and appreciate the construction of recovery centers.

# 5 Conclusions and Future Work

In this paper, we propose OptiChat to address the common challenge that end users of optimization models are often not optimization experts. By leveraging the recent advancements in LLMs, OptiChat bypasses the need for inefficient back-and-forth coordination with optimization experts. Through straightforward natural language conversations with models, the practitioners can benefit from autonomous and instant responses. The integration of optimization tools with LLMs is demonstrated to be essential for providing a richer and more reliable context for analysis, rather than relying entirely on natural language processing to generate plausible explanations.

Although OptiChat, at its current status, does not achieve perfect accuracy in executing tools or generating code, it is anticipated that this issue can be further mitigated by foreseeable advancements in LLM technology and targeted supervised fine-tuning specific to optimization tasks. One important step in this direction is the development of a larger and more diverse testing dataset to better evaluate and improve OptiChat's performance. Data augmentation techniques, such as prompting the LLM to rephrase the same questions, can be incorporated. Furthermore, users may exhibit varied preferences for answers and explanations, depending on their background and application context, especially in multi-turn conversations. To accommodate this, recent progress in reinforcement learning from human feedback (RLHF) offers a promising pathway. By training a reward model based on user preferences, we can better align OptiChat's responses with the diverse expectations of its users. A more robust method can also be developed to evaluate users' satisfaction with OptiChat in multi-turn conversations. Last but not least, more explainable optimization techniques, such as inherently interpretable policies in the form of decision trees [31, 32], can be added to the OptiChat. Thanks to OptiChat's modular design, incorporating new explanation strategies or toolsets can be achieved with minimal overhead. We believe that OptiChat, along with the new dataset and the insights into query classification, could serve as a catalyst in this understudied area and draw more attention from the optimization and the machine learning community.

# Appendix A Dataset Summary

Table A1 summarizes the collection of optimization models used in our study by type, size, and application domain. The dataset spans 24 models in total with medium sizes.

For context, NLP4LP [18], a dataset in the similar field, contains 355 instances in total (18 MILP instances), with parameter statistics of 3 / 8 / 18 (Min./Avg./Max.).

**Table A1** Summary of Optimization Models by Type, Size, and Domain.

| Category | Count |
|---|---|
| # Total Models | 24 |
| # LP | 10 |
| # MILP | 14 |
| # Parameters (Min./Avg./Max.) | 21 / 1,264 / 25,432 |
| # Variables (Min./Avg./Max.) | 14 / 1,002 / 14,880 |
| # Constraints (Min./Avg./Max.) | 10 / 321 / 2,672 |
| Manufacturing & Scheduling | 8 |
| Supply Chain & Logistics | 8 |
| Chemical & Process Engineering | 4 |
| Power & Energy | 3 |
| Other | 1 |

# Appendix B Qualitative Evaluation of Model Descriptions

The quality of the model descriptions was assessed by expert participants. Each participant was provided with optimization models and asked to review the underlying code before comparing their own understanding with the descriptions generated by OptiChat. The evaluation considered several key aspects: (i) accuracy, captured by the presence of obvious errors or deviations from the actual meaning of the model; (ii) breadth, reflecting whether the description covered the important components of the model rather than focusing only on a narrow subset and (iii) clarity, indicating whether the description was approachable and would be easy for non-expert readers to understand. These criteria together served as the evaluation metric for determining the overall quality of the model descriptions.

# Appendix C Prompts

```
component_interpretation_prompt = """You are an operations
research expert and your role is to use PLAIN ENGLISH to
interpret an optimization model written in Pyomo. The Pyomo code
is given below:
-----
{code}
-----
Here are the name of {component_type} that need to be described
-----
{component_names}
-----
Your task is carefully inspect the code and write a description
for each of the components.
Then, generate a json file accordingly with the following format
(STICK TO THIS FORMAT!)
{model_interpretation_json}
- description should be either physical meanings, intended use,
or any other relevant information about the component.
- Generate the complete json file and don't omit anything.
- Use 'name' and 'description' as the keys, and provide the name
and description of the component as the values."""

model_illustration_prompt = """You are an operations research
expert and your role is to introduce an optimization model to non
-experts, based on an abstract representation of the model in
json format.
The json representation is given below:
-----
{json_representation}
-----
- Start with a brief introduction of the model, what the problem
is about, who is using the model, and what the model is trying to
 achieve.
- Explain what decisions (variables) are to be made
- Explain what data or information (parameters) is already known
- Explain what constraints are imposed on the decisions
- Explain what the objective is, what is being optimized
The explanation must be coherent and easy to understand for the
users who are experts in the filed for which this model is built
but not in optimization."""

iis_inference_prompt = """You are an operations research expert
and your role is to infer why an optimization model is infeasible
, based on an abstract representation of the infeasible model in
json format.
```

```
Particularly, your team has identified the Irreducible Infeasible
 Subset (IIS) of the model, which is given below:
-----
{iis_info}
-----
To understand what the parameters and the constraints mean, the
json representation is given below for your reference:
-----
{json_representation}
-----
- Introduce to the user what constraints are potentially causing
the infeasibility, and what parameters are involved in these
constraints.
- Explain the relationship between the constraints and the
parameters, and infer why the constraints are conflicting with
each other.
- Provide inference by analyzing their physical meanings, and
AVOID using jargon and symbols as much as possible but the
explanation style must be formal.
- Recommend some parameters that you believe can be adjusted to
make the model feasible.
- Parameters recommended for adjustment MUST be changeable
physically in practice. For example, molecular weight of a
molecule is not changeable in practice.
- Assess the practical implications of the recommendations. For
example, increasing the number of workers implies hiring more
workers, which incurs additional costs."""

coordinator_prompt = """You're a coordinator in a team of
optimization experts. The goal of the team is to help non-experts
 analyze an optimization problem. Your task is to choose the next
 expert to work on the problem based on the current situation.
Here's the list of agents in your team:
-----
{agents}
-----
Considering the conversation, generate a json file with the
following format:
{{ "agent_name": "Name of the agent you want to call next", "task
": "The task you want the agent to carry out" }}
to identify the next agent to work on the problem, and also the
task it has to carry out.
- Only generate the json file, and don't generate any other text.
- DO NOT change the keys of the json file, only change the values
. Keys are "agent_name" and "task".
- if you think the problem is solved, generate the json file
below:
{{ "agent_name": "Explainer", "task": "DONE" }}"""
```

```
explainer_prompt = """You're an optimization expert who helps
your team answer user queries in MARKDOWN format.
- The users are not experts in optimization, but they are experts
 in the filed for which this model is built.
- Provide a detailed explanation only when you believe the users
need more context about optimization to understand your
explanation.
- Otherwise, the explanation must be succinct and concise,
because users may be distracted by too much information.
- If Operators and Programmers in your team have provided
technical feedback, then you need to summarize the feedback
because the user cannot see them."""

syntax_reminder_prompt = """You're an operator working on a pyomo
 model.
Your task is to identify the following arguments:
- the component names that the user is interested in,
- the most appropriate function that can answer the user's query,
- the model that the user is querying.
then call the predefined syntax_guidance function to generate
syntax guidance.
----- Instruction to select the most appropriate function -----
you MUST select a function from ```{function_names}```, DO NOT
make up your own function.
1. feasibility_restoration:
Use when: The model is infeasible and you need to find out the
minimal change to specific [component name] for restoring
feasibility.
Example: "How much should we adjust the [component name] to make
the model feasible"
Example: "I believe changing [component name] is practical, by
how much do I need to change in order to make the model feasible"
[component name] category: parameters. If only constraint name is
 provided in the query, you need to infer the parameters involved
 in the constraint.

2. components_retrival:
Use when: You need to know the current values or expressions of [
component name] within the model.
Example: "What are the values of the [component name]"
Example: "How many [component name] are currently available"
[component name] category: sets, parameters, variables,
constraints, or objectives.

3. sensitivity_analysis:
Use when: The model is feasible and you want to understand the
impact of changing [component name] on the optimal objective
value, **without specifying the extent of changes**.
Example: "How will the optimal profit change with the change in
the [component name]" (didn't specify how much the change is)
```

```
Example: "How stable is the objective value in response to
variations in the [component name]" (didn't specify how much the
change is)
Example: "Will the optimal value be greatly affected if we have
more [component name]" (didn't specify how much the change is)
[component name] category: parameters.

4. evaluate_modification:
Use when: The model is feasible and you want to understand the
impact of changing [component name] on the optimal objective
value, **by specifying the extent of changes**.
Example: "How will the optimal profit change with **a 10%
increase** in the [component name]" (specified the change is **a
10% increase**)
Example: "How stable is the objective value in response to the
modification that [component name] is **decreased by 20 units**"
(specified the change is **decreased by 20 units**)
Example: "Will the optimal value be greatly affected if we have
**two more** [component name]" (specified the change is **two
more**)
[component name] category: parameters or variables.

5. external_tools:
Use when: User doubts the model's optimal solution and provides a
 counterexample, and you want to add new constraints to implement
 the counterexample.
Example: "Why is it not recommended to have [component name]
lower than 400 in the optimal solution"
Example: "Why isn't [component name] and [component name] both
used in the optimal scenario"
[component name] category: parameters or variables.

----- Instruction to determine the correct component name -----
The [component name] MUST be in a symbolic form, instead of its
description.
Use the following dictionary to find the correct [component name]
 based on its description:
{component_name_meaning_pairs}

----- Instruction to find the queried model -----
In the form of 'model_integer', e.g. 'model_1'"""

# all necessary information has been provided
operator_prompt = """You're an optimization expert who helps your
 team to access and interact with optimization models by internal
 tools.
Your task is to invoke the most appropriate tool correctly based
on the user's query and system reminders."""
```

```
code_reminder_prompt = """{source_code}\n# YOUR CODE GOES HERE\n
"""

programmer_prompt = """You're an optimization expert who helps
your team to write pyomo code to answer users questions, such as
- write code snippet to revise the model, only when the user
doubts the model's optimal solution and provides a counterexample
- write code snippet to print out the information useful for
answering the user's question
Output Format:
==========
'''python
YOUR CODE SNIPPET
'''
==========
Here are some example questions and their answer codes:
----- EXAMPLE 1 -----
Question: Why is it not recommended to use just one supplier for
roastery 2?

Answer Code:
'''python
# user is actually interested in the case that only one supplier
can supply roastery 2 and does not believe the optimal solution.
model.force_one_supplier = ConstraintList()
model.force_one_supplier.add(sum(model.z[s,'roastery2'] for s in
model.suppliers) <= 1)
for s in model.suppliers:
    model.force_one_supplier.add(model.x[s,'roastery2'] <= model.
    capacity_in_supplier[s] * model.z[s, 'roastery2'])
    from pyomo.environ import SolverFactory, TerminationCondition

# standard code to solve the model. Don't change this code if you
 need to solve a mode.
solver = SolverFactory('gurobi')  # only gurobi is available in
env
solver.options['TimeLimit'] = 300  # 5min time limit
results = solver.solve(model, tee=False)  # tee must be False to
suppress solver output, otherwise the output is overwhelming
print("Solver Status: ", results.solver.status)
print("Termination Condition: ", results.solver.
termination_condition)
# always check the termination condition and optimal objective
value first
if results.solver.termination_condition == TerminationCondition.
optimal:
    from pyomo.environ import Objective
    from pyomo.environ import value
    for obj_name, obj in model.component_map(Objective).items():
        print('Optimal Objective Value: ', value(obj))
```

```
else:
    print("Model is infeasible or unbounded, no optimal objective
     value is available.")

# I print out the new optimal objective value so that you can
tell the user how the objective value changes if only one
supplier supplies roastery 2.
print('If forcing only one supplier to supply roastery 2, the
optimal objective value will become: ', model.obj())
```

```
----- EXAMPLE 2 -----
Question: Why is it not recommended to have production cost
larger than transportation cost in the optimal setting?

Answer Code:
```python
# user does not believe the optimal solution obtained when
production cost smaller than transportation cost.
# so we force production cost to be less than transportation cost
 to see what will happen.
model.counter_example = ConstraintList()
model.counter_example.add(model.production <= model.
transportation)

# standard code to solve the model. Don't change this code if you
 need to solve a mode.
solver = SolverFactory('gurobi')  # only gurobi is available in
env
solver.options['TimeLimit'] = 300  # 5min time limit
results = solver.solve(model, tee=False)  # tee must be False to
suppress solver output, otherwise the output is overwhelming
print("Solver Status: ", results.solver.status)
print("Termination Condition: ", results.solver.
termination_condition)
# always check the termination condition and optimal objective
value first
if results.solver.termination_condition == TerminationCondition.
optimal:
    from pyomo.environ import Objective
    from pyomo.environ import value
    for obj_name, obj in model.component_map(Objective).items():
        print('Optimal Objective Value: ', value(obj))
else:
    print("Model is infeasible or unbounded, no optimal objective
     value is available.")

# I print out the new optimal objective value so that you can
tell the user how the objective value changes.
```
```

```
print('If forcing production cost be smaller than transportation
cost, the optimal objective value will become: ', model.obj())
```
- Code reminder has provided you with the source code of the
pyomo model
- Your written code will be added to the lines with substring: "#
 YOUR CODE GOES HERE" So, you don't need to repeat the source
code that has already been provided by Code reminder.
- The standard code for re-solving the model has been given in
the examples,
So, you MUST use the standard code to re-solve the model to avoid
 undesired execution errors and long execution result.
- Your written code should be accompanied by comments to explain
the purpose of the code.
- Evaluator will execute the new code for you and read the
execution result.
So, you MUST print out the model information that you believe is
necessary for the user's question."""

evaluator_prompt = """You're an optimization expert who helps
your team to review pyomo code,
based on the execution result of the code provided by the
programmer.
Is the code bug-free and valid to answer the user's query?
Generate the following json file if you accept the code, and
provide your own comment.{{ "decision": "accept", "comment": "
your own comment" }}
Generate the following json file if you reject the code, and
provide your own comment. {{ "decision": "reject", "comment": "
your own comment" }}
- Only generate the json file, and don't generate any other text.
- Use 'decision' and 'comment' as the keys,
- choose 'accept' or 'reject' for the decision, and provide your
own comment.
- Note that infeasibility caused by the new constraints may be
acceptable.
This is because programmers are trying to create a counterfactual
 example that the user is interested in, and this counterfactual
example may be infeasible in nature."""

# Appendix D Workflow Example

Figure D1, Figure D2, and Figure D3 show the workflow using an example query: "what if I change maximum normal production at facility two significantly, say increase it by 20 units, what will the profit be?" This query is first classified as solution-specific and delegated from the Coordinator to the Engineer. Among the Engineer's sub-agents, the reminder identifies that the appropriate explanation strategy for this what-if query is to evaluate the modification. The model information, previously pre-processed by the Illustrator, enables the reminder to correctly associate the term "normal production" with the parameter acronym "pdf". Based on this, the reminder invokes a predefined function to generate syntax guidance tailored to the "evaluate modification" function and the "pdf" component, capturing details such as the dimension and pattern of its indexes. This syntax guidance, along with the query, is then passed to the Operator, who formally invokes "evaluate modification" with the correct function arguments and computes the solution. Finally, the Explainer communicates the result to the user with context-aware natural language.

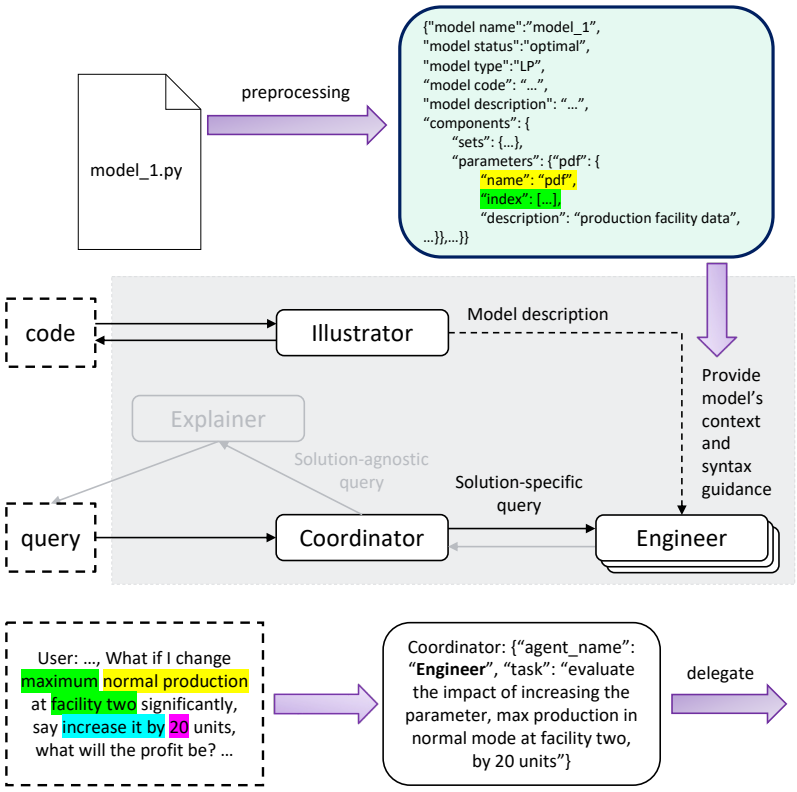


**Fig. D1** An Illustrative example of the what-if query answered by the workflow (a)

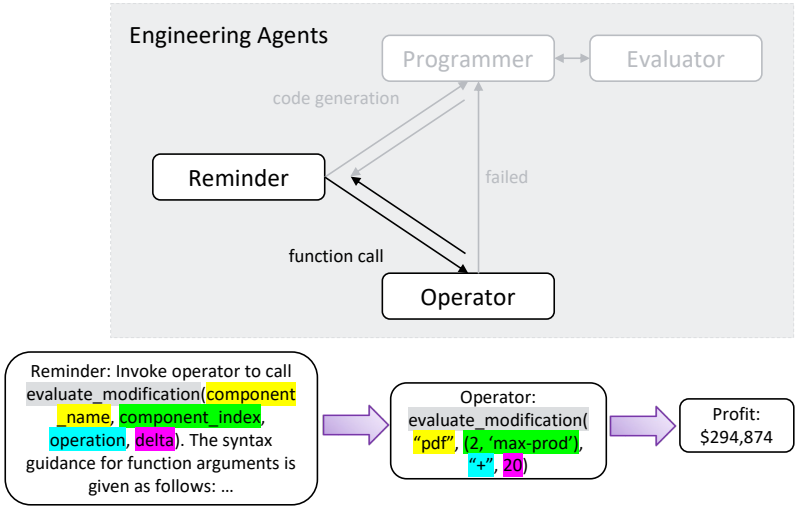


**Fig. D2** An Illustrative example of the what-if query answered by the workflow (b)

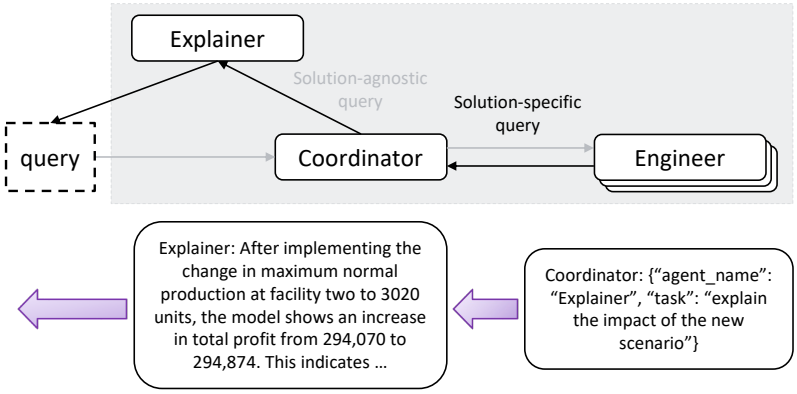


**Fig. D3** An Illustrative example of the what-if query answered by the workflow (c)

# Appendix E Example Conversation On User Interface

Figure E4 and Figure E5 present two representative conversations for the diagnosing and what-if queries.

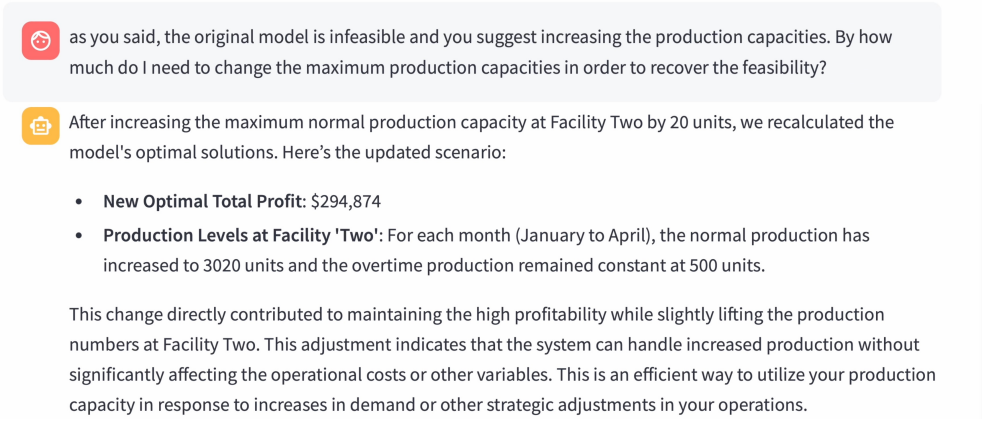


**Fig. E4** A conversation example of the diagnosing query answered by OptiChat

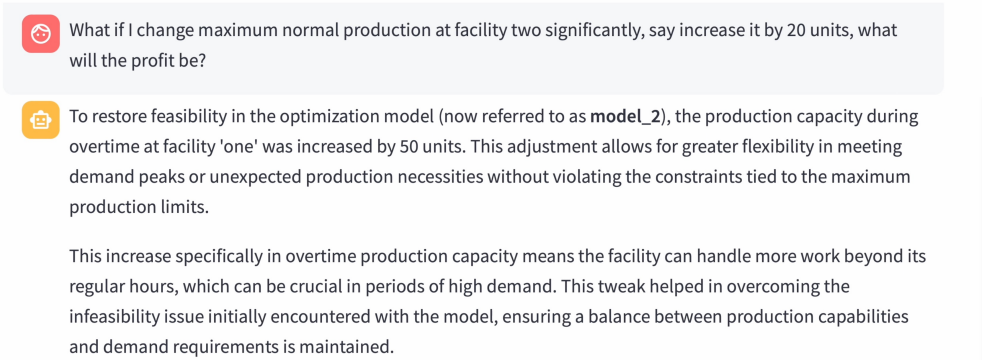


**Fig. E5** A conversation example of the what-if query answered by OptiChat

# References

[1] OpenAI: Gpt-4 technical report (2023)

[2] Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., Rodriguez, A., Joulin, A., Grave, E., Lample, G.: LLaMA: Open and Efficient Foundation Language Models (2023). https://arxiv.org/abs/2302.13971

[3] Bran, A.M., Cox, S., Schilter, O., Baldassari, C., White, A.D., Schwaller, P.: Augmenting large language models with chemistry tools. Nature Machine Intelligence **6**, 525–535 (2024) https://doi.org/10.1038/s42256-024-00832-8

[4] Jablonka, K.M., Schwaller, P., Ortega-Guerrero, A., Smit, B.: Leveraging large language models for predictive chemistry. Nature Machine Intelligence **6**, 161–169

(2024) https://doi.org/10.1038/s42256-023-00788-1

[5] Boiko, D.A., MacKnight, R., Kline, B., Gomes, G.: Autonomous chemical research with large language models. Nature **624**, 570–578 (2023) https://doi.org/10.1038/s41586-023-06792-0

[6] Li, Y., Gao, C., Song, X., Wang, X., Xu, Y., Han, S.: Druggpt: A gpt-based strategy for designing potential ligands targeting specific proteins. bioRxiv (2023) https://doi.org/10.1101/2023.06.29.543848

[7] Lin, Z., Akin, H., Rao, R., Hie, B., Zhu, Z., Lu, W., Smetanin, N., Verkuil, R., Kabeli, O., Shmueli, Y., Santos Costa, A., Fazel-Zarandi, M., Sercu, T., Candido, S., Rives, A.: Evolutionary-scale prediction of atomic-level protein structure with a language model. Science **379**, 1123–1130 (2023) https://doi.org/10.1126/science.ade2574

[8] Luo, R., Sun, L., Xia, Y., Qin, T., Zhang, S., Poon, H., Liu, T.-Y.: Biogpt: generative pre-trained transformer for biomedical text generation and mining. Briefings in Bioinformatics **23**, 409 (2022) https://doi.org/10.1093/bib/bbac409

[9] Thawkar, O., Shaker, A., Mullappilly, S.S., Cholakkal, H., Anwer, R.M., Khan, S., Laaksonen, J., Khan, F.S.: XrayGPT: Chest Radiographs Summarization using Medical Vision-Language Models (2023). https://arxiv.org/abs/2306.07971

[10] Sallam, M.: Chatgpt utility in healthcare education, research, and practice: Systematic review on the promising perspectives and valid concerns. Healthcare **11** (2023) https://doi.org/10.3390/healthcare11060887

[11] Dowling, M., Lucey, B.: Chatgpt for (finance) research: The bananarama conjecture. Finance Research Letters **53**, 103662 (2023) https://doi.org/10.1016/j.frl.2023.103662

[12] Wu, S., Irsoy, O., Lu, S., Dabravolski, V., Dredze, M., Gehrmann, S., Kambadur, P., Rosenberg, D., Mann, G.: BloombergGPT: A Large Language Model for Finance (2023). https://arxiv.org/abs/2303.17564

[13] Lopez-Lira, A., Tang, Y.: Can ChatGPT Forecast Stock Price Movements? Return Predictability and Large Language Models (2024). https://arxiv.org/abs/2304.07619

[14] Wang, X., Anwer, N., Dai, Y., Liu, A.: Chatgpt for design, manufacturing, and education. Procedia CIRP **119**, 7–14 (2023) https://doi.org/10.1016/j.procir.2023.04.001

[15] Badini, S., Regondi, S., Frontoni, E., Pugliese, R.: Assessing the capabilities of chatgpt to improve additive manufacturing troubleshooting. Advanced Industrial and Engineering Polymer Research **6**, 278–287 (2023) https://doi.org/10.1016/j.

aiepr.2023.03.003

[16] Li, B., Mellou, K., Zhang, B., Pathuri, J., Menache, I.: Large Language Models for Supply Chain Optimization (2023). https://arxiv.org/abs/2307.03875

[17] AhmadiTeshnizi, A., Gao, W., Brunborg, H., Talaei, S., Udell, M.: OptiMUS-0.3: Using Large Language Models to Model and Solve Optimization Problems at Scale (2024). https://arxiv.org/abs/2407.19633

[18] Ahmaditeshnizi, A., Gao, W., Udell, M.: OptiMUS: Scalable optimization modeling with (MI)LP solvers and large language models. In: Salakhutdinov, R., Kolter, Z., Heller, K., Weller, A., Oliver, N., Scarlett, J., Berkenkamp, F. (eds.) Proceedings of the 41st International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 235, pp. 577–596. PMLR, Vienna, Austria (2024). https://proceedings.mlr.press/v235/ahmaditeshnizi24a.html

[19] AhmadiTeshnizi, A., Gao, W., Udell, M.: OptiMUS: Optimization Modeling Using MIP Solvers and large language models (2023). https://arxiv.org/abs/2310.06116

[20] Xiao, Z., Zhang, D., Wu, Y., Xu, L., Wang, Y.J., Han, X., Fu, X., Zhong, T., Zeng, J., Song, M., Chen, G.: Chain-of-experts: When LLMs meet complex operations research problems. In: The Twelfth International Conference on Learning Representations (2024). https://openreview.net/forum?id=HobyL1B9CZ

[21] Tang, Z., Huang, C., Zheng, X., Hu, S., Wang, Z., Ge, D., Wang, B.: ORLM: Training Large Language Models for Optimization Modeling (2024). https://arxiv.org/abs/2405.17743

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

[23] Slack, D., Krishna, S., Lakkaraju, H., Singh, S.: Explaining machine learning models with interactive natural language conversations using talktomodel. Nature Machine Intelligence **5**, 873–883 (2023) https://doi.org/10.1038/s42256-023-00692-8

[24] Ribeiro, M.T., Singh, S., Guestrin, C.: "why should i trust you?": Explaining the predictions of any classifier. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. KDD '16, pp. 1135–1144. Association for Computing Machinery, New York, NY, USA (2016). https://doi.org/10.1145/2939672.2939778

[25] Lundberg, S.M., Lee, S.-I.: A unified approach to interpreting model predictions.

In: Proceedings of the 31st International Conference on Neural Information Processing Systems. NIPS'17, pp. 4768–4777. Curran Associates Inc., Red Hook, NY, USA (2017)

[26] Selvaraju, R.R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., Batra, D.: Grad-cam: Visual explanations from deep networks via gradient-based localization. In: 2017 IEEE International Conference on Computer Vision (ICCV), pp. 618–626 (2017). https://doi.org/10.1109/ICCV.2017.74

[27] Karimi, A.-H., Barthe, G., Balle, B., Valera, I.: Model-agnostic counterfactual explanations for consequential decisions. In: Chiappa, S., Calandra, R. (eds.) Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics. Proceedings of Machine Learning Research, vol. 108, pp. 895–905. PMLR, Palermo, Sicily, Italy (2020). https://proceedings.mlr.press/v108/karimi20a.html

[28] Rardin, R.L.: Optimization in Operations Research, 2nd edn. Pearson Education, Hoboken, NJ (2016)

[29] Greenberg, H.: A functional description of analyze: A computer-assisted analysis system for linear programming models. ACM Transactions on Mathematical Software **9**, 18–56 (1983) https://doi.org/10.1145/356022.356024

[30] Greenberg, H.J.: Analyze: A computer-assisted analysis system for linear programming models. Operations Research Letters **6**, 249–255 (1987) https://doi.org/10.1016/0167-6377(87)90057-5

[31] Bertsimas, D., Stellato, B.: The voice of optimization. Machine Learning **110**(2), 249–277 (2021) https://doi.org/10.1007/s10994-020-05893-5

[32] Goerigk, M., Hartisch, M.: A framework for inherently interpretable optimization models. European Journal of Operational Research **310**(3), 1312–1324 (2023) https://doi.org/10.1016/j.ejor.2023.04.013

[33] Lumbreras, S., Tejada, D., Elechiguerra, D.: Explaining the solutions of the unit commitment with interpretable machine learning. International Journal of Electrical Power & Energy Systems **160**, 110106 (2024) https://doi.org/10.1016/j.ijepes.2024.110106

[34] Čyras, K., Lee, M., Letsios, D.: Schedule explainer: An argumentation-supported tool for interactive explanations in makespan scheduling. In: International Workshop on Explainable, Transparent Autonomous Agents and Multi-agent Systems, pp. 243–259 (2021)

[35] Čyras, K., Letsios, D., Misener, R., Toni, F.: Argumentation for explainable scheduling. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 33, pp. 2752–2759 (2019)

[36] Collins, A., Magazzeni, D., Parsons, S.: Towards an argumentation-based approach to explainable planning. In: ICAPS 2019 Workshop XAIP Program Chairs, p. 16 (2019)

[37] Forel, A., Parmentier, A., Vidal, T.: Explainable Data-Driven Optimization: From Context to Decision and Back Again. In: Krause, A., Brunskill, E., Cho, K., Engelhardt, B., Sabato, S., Scarlett, J. (eds.) Proceedings of the 40th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 202, pp. 10170–10187. PMLR, Honolulu, HI, USA (2023). https://proceedings.mlr.press/v202/forel23a.html

[38] Ji, Z., Lee, N., Frieske, R., Yu, T., Su, D., Xu, Y., Ishii, E., Bang, Y.J., Madotto, A., Fung, P.: Survey of hallucination in natural language generation. ACM Comput. Surv. **55**(12) (2023) https://doi.org/10.1145/3571730

[39] Chen, H., Constante-Flores, G.E., Li, C.: Diagnosing infeasible optimization problems using large language models. INFOR: Information Systems and Operational Research **62**(4), 1–15 (2024) https://doi.org/10.1080/03155986.2024.2385189

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

[41] Huang, C., Tang, Z., Hu, S., Jiang, R., Zheng, X., Ge, D., Wang, B., Wang, Z.: Orlm: A customizable framework in training large models for automated optimization modeling. Operations Research (2025)

[42] Astorga, N., Liu, T., Xiao, Y., Schaar, M.: Autoformulation of Mathematical Optimization Models Using LLMs (2025). https://arxiv.org/abs/2411.01679

[43] Yang, Z., Wang, Y., Huang, Y., Guo, Z., Shi, W., Han, X., Feng, L., Song, L., Liang, X., Tang, J.: OptiBench Meets ReSocratic: Measure and Improve LLMs for Optimization Modeling (2025). https://arxiv.org/abs/2407.09887

[44] Romera-Paredes, B., Barekatain, M., Novikov, A., Balog, M., Kumar, M.P., Dupont, E., Ruiz, F.J.R., Ellenberg, J.S., Wang, P., Fawzi, O., Kohli, P., Fawzi, A.: Mathematical discoveries from program search with large language models. Nature **625**, 468–475 (2024) https://doi.org/10.1038/s41586-023-06924-6

[45] Stein, N., Vermetten, D., Bäck, T.: In-the-loop hyper-parameter optimization for llm-based automated design of heuristics. ACM Trans. Evol. Learn. Optim. (2025) https://doi.org/10.1145/3731567

[46] Lawless, C., Schoeffer, J., Le, L., Rowan, K., Sen, S., St. Hill, C., Suh, J., Sarrafzadeh, B.: "i want it that way": Enabling interactive decision support using large language models and constraint programming. ACM Trans. Interact. Intell. Syst. **14**(3) (2024) https://doi.org/10.1145/3685053

[47] JU, D., Jiang, S., Cohen, A., Foss, A., Mitts, S., Zharmagambetov, A., Amos, B., Li, X., Kao, J.T., Fazel-Zarandi, M., Tian, Y.: To the Globe (TTG): Towards Language-Driven Guaranteed Travel Planning (2024). https://arxiv.org/abs/2410.16456

[48] Kikuta, D., Ikeuchi, H., Tajiri, K., Nakano, Y.: Routeexplainer: an explanation framework for vehicle routing problem. In: Pacific-asia Conference on Knowledge Discovery and Data Mining (2024)

[49] Gurobi Optimization, LLC: Gurobi Optimizer Reference Manual. (2022). https://www.gurobi.com

[50] Chinneck, J.W., Dravnieks, E.W.: Locating minimal infeasible constraint sets in linear programs. ORSA Journal on Computing **3**, 157–168 (1991) https://doi.org/10.1287/ijoc.3.2.157

[51] Tamiz, M., Mardle, S.J., Jones, D.F.: Detecting iis in infeasible linear programmes using techniques from goal programming. Computers & Operations Research **23**, 113–119 (1996) https://doi.org/10.1016/0305-0548(95)00018-H

[52] Guieu, O., Chinneck, J.W.: Analyzing infeasible mixed-integer and integer linear programs. INFORMS Journal on Computing **11**, 63–77 (1999) https://doi.org/10.1287/ijoc.11.1.63

[53] IBM: IBM ILOG CPLEX 22.1.0 User's Manual (2022). https://www.ibm.com/docs/en/icos/22.1.0?topic=optimizers-users-manual-cplex

[54] Chinneck, J.W.: Feasibility and Infeasibility in Optimization vol. 118. Springer, New York, NY (2008). https://doi.org/10.1007/978-0-387-74932-7

[55] Bertsimas, D., Tsitsiklis, J.: Introduction to Linear Optimization, 1st edn. Athena Scientific, Nashua, NH, USA (1997)

[56] MOSEK ApS: MOSEK Optimizer API for Python 10.1.8. (2023). https://docs.mosek.com/latest/capi/index.html

[57] Wu, Q., Bansal, G., Zhang, J., Wu, Y., Li, B., Zhu, E., Jiang, L., Zhang, X., Zhang, S., Liu, J., Awadallah, A.H., White, R.W., Burger, D., Wang, C.: AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation (2023). https://arxiv.org/abs/2308.08155