



IE 492 GRADUATION PROJECT

FINAL REPORT

Conversational Schedule Explainer with ChatGPT

Murat Tutar

Project Supervisors: Ali Tamer Ünal, Zeki Caner Taşkın

Abstract

This paper is a final report on the project "Conversational Schedule Explainer with ChatGPT." It addresses the challenge of transparency and understanding in employee scheduling in the service industry due to the continuous and fluctuating nature of demand. The project integrates advanced optimization models with a user-friendly interface, utilizing Gurobi for optimization and ChatGPT API for natural language processing. This system aims to provide clear explanations for scheduling decisions, improving employee engagement and operational efficiency. The report also delves into the underlying functions of the system, exploring how it answers complex scheduling queries and facilitates user interactions through an intuitive interface. It emphasizes the importance of this ergonomic design for a better user experience in workforce management.

Özet

Bu belge, "Conversational Schedule Explainer with ChatGPT" adlı projenin final raporudur. Özellikle hizmet sektöründe çalışanların periyodik çalışma takvimleri konusunda şeffaflık ve anlaşılırlığı artırmayı ele alır. Proje, kullanıcı dostu bir arayüz ile gelişmiş optimizasyon modellerini entegre eder, optimizasyon için Gurobi'yi ve doğal dil işleme için ChatGPT API'sini kullanır. Bu sistem, çalışma takvimi kararları için net açıklamalar sağlayıcı, çalışan katılımını ve operasyonel verimliliği iyileştirmeyi amaçlamaktadır. Rapor ayrıca, karmaşık programlama yöntemleriyle sorulara nasıl cevap verdiği ve sezgisel bir arayüz aracılığıyla kullanıcı etkileşimlerini nasıl kolaylaştırdığını inceleyen sisteme de temel işlevlere de derinlemesine iner. İşgücü yönetiminde daha iyi bir kullanıcı deneyimi için bu ergonomik tasarımın önemini vurgular.

Table of Contents

1.	Introduction.....	4
2.	Problem Definition, Requirements, and Limitations.....	5
2.1.	Problem Definition.....	5
2.2.	Requirements.....	6
2.3.	Limitations.....	7
2.4.	Context Diagram.....	7
3.	Analysis for Solution/Design Methodology.....	8
3.1.	Literature Review.....	8
3.2.	Alternative Solution/Design Approaches.....	10
3.3.	Assumptions.....	11
3.4.	Brief Overview of the Selected Approach.....	12
3.5.	IE Tools & Methods Integration.....	13
4.	Development of Solutions & Comparison of Alternatives & Recommendation.....	14
4.1.	Detailed Technical Description of the Design Process.....	14
4.1.1.	List Of Frequently Asked Questions.....	17
4.1.2.	Answering Query Questions	18
4.1.3.	Answering Request Questions.....	20
4.1.4.	Answering Reasoning Questions.....	23
4.2.	Creation of User Interface.....	26
4.3.	Integration of ChatGPT.....	27
4.4.	Comparison and Evaluation.....	27
5.	Suggestions for Successful Implementation.....	28
6.	Conclusions and Discussion.....	30
7.	References.....	32
8.	Appendix.....	33

1. Introduction

Employee scheduling in the service industry, characterized by dynamic and complex operational demands, presents a critical challenge that significantly impacts both staff satisfaction and business efficiency. This project addresses the inherent opacity in conventional scheduling systems, which often leaves employees seeking greater clarity and rationale behind their assigned shifts and tasks.

The primary objective of this project is to revolutionize employee scheduling in the service industry by enhancing transparency and understanding. Traditional scheduling systems often lead to confusion and dissatisfaction among staff due to their opaque nature. Our goal is to address this challenge by integrating advanced optimization models with a user-friendly interface for an exemplary user experience. This integration, leveraging Gurobi for optimization and ChatGPT API for natural language processing, aims to create a system that communicates the reasoning behind the scheduling decisions in a clear and accessible manner.

This project goes beyond the technical aspects of scheduling; it delves into the human element of workforce management. By enabling employees to interactively query and understand their schedules, we foster a more engaged, informed, and satisfied workforce. This is crucial in the service industry, where employee morale directly influences customer experience and business performance.

Furthermore, the project significantly reduces the administrative burden associated with managing and explaining schedules, thus streamlining operational processes. By leveraging AI, we offer a scalable, efficient solution that can adapt to varying demands and

complexities typical in service environments, including the case study of airport staff scheduling.

In summary, this project represents a significant advancement in employee scheduling systems. It demonstrates how the integration of optimization models and AI can enhance transparency, improve employee engagement, and maintain operational efficiency in the service industry. This report will explore in detail the problem definition, solution approach, implementation, and the broader impacts of this innovative endeavor.

2. Problem Definition, Requirements, and Limitations

2.1. Problem Definition

The fundamental challenge this project addresses is the lack of transparency in employee scheduling within the service industry. Employee scheduling, especially in environments with fluctuating demands and flexible employee availability, is a complex task. Planners need to balance employee preferences, business constraints, and variable demand. However, in today's business environment things do not end up here. Despite achieving an optimum schedule, employees often seek to understand the logic behind their schedules and the feasibility of potential schedule-related requests. This need for explanation and clarity becomes more pronounced in settings like the service industry, where schedules are inherently complex due to continuous and changing demands.

While our project uses airport staff scheduling as a case study, the problem and its solutions are broadly applicable across the service industry. The generated staff schedule serves as a medium to perform the central aspect of our project: explaining a given schedule. We've implemented functions to answer frequently asked questions (FAQs) from employees, using model constraints and objectives for explanations. The project encompasses

constructing a schedule, implementing FAQ response functions, and integrating ChatGPT API for natural language processing and user interaction.

Briefly, this project stands at the intersection of advanced scheduling requirements and the transformative potential of explainable AI. By utilizing efficient search algorithms, optimization models, and providing comprehensible explanations, we aim to usher in a new paradigm in workforce management. This approach not only enhances employee engagement but also provides managers with valuable insights for continuous improvement.

2.2. Requirements

The project has key requirements for success. From a technical perspective, the system must prioritize user-friendliness with an intuitive interface. It should generate clear explanations for schedule assignments using ChatGPT API, integrate Gurobi for optimized schedules, and be scalable to adapt to changing demands.

User-oriented requirements emphasize clear and comprehensive explanations for schedule assignments and support for interactive querying. User satisfaction is crucial and requires ongoing feedback and usability testing.

Ethical and legal considerations involve data confidentiality and privacy measures, compliance with labor laws, and ethical AI usage by training models on ethical guidelines and addressing bias and discrimination.

In summary, these requirements guide the development of a transparent and interactive scheduling system that enhances employee engagement, ensures operational efficiency, and upholds ethical standards in workforce management.

2.3. Limitations

The project navigates several limitations, including computational challenges in re-optimizing schedules for large problem sizes due to the NP-Hardness and the inability to address certain types of queries due to model limitations or the lack of clarity in the nature of the questions.

Additionally, the ethical aspect of the project is critical. It involves teaching the model about data confidentiality, ensuring consent for sharing schedule information, and adhering to privacy laws. The system's design and operation must comply with relevant labor laws and industry regulations, which vary by region and work type.

2.4. Context Diagram

There are two main stakeholders: Employees and Managers. Each has different but interrelated benefits. Both of them are connected to the system by prompting questions regarding the schedule. Then, utilizing ChatGPT, the system gets the question and maps it to a related function which will provide an explanatory answer as an output and ChatGPT takes that output and converts this to an understandable answer for the user.

Each stakeholder benefits from the system in different ways after successful implementation:

- Employees will be happier and have an increased job satisfaction level since they will be sure that the scheduling decisions are made based on a logic considering the same rules for everyone with a transparent objective and without any kinds of nepotism.
- Managers will be happier because motivated employees will be more productive, making the company more profitable. Additionally, they can also get a good

understanding of the overall scheduling model and focus on areas of improvement for more efficient schedules in future.

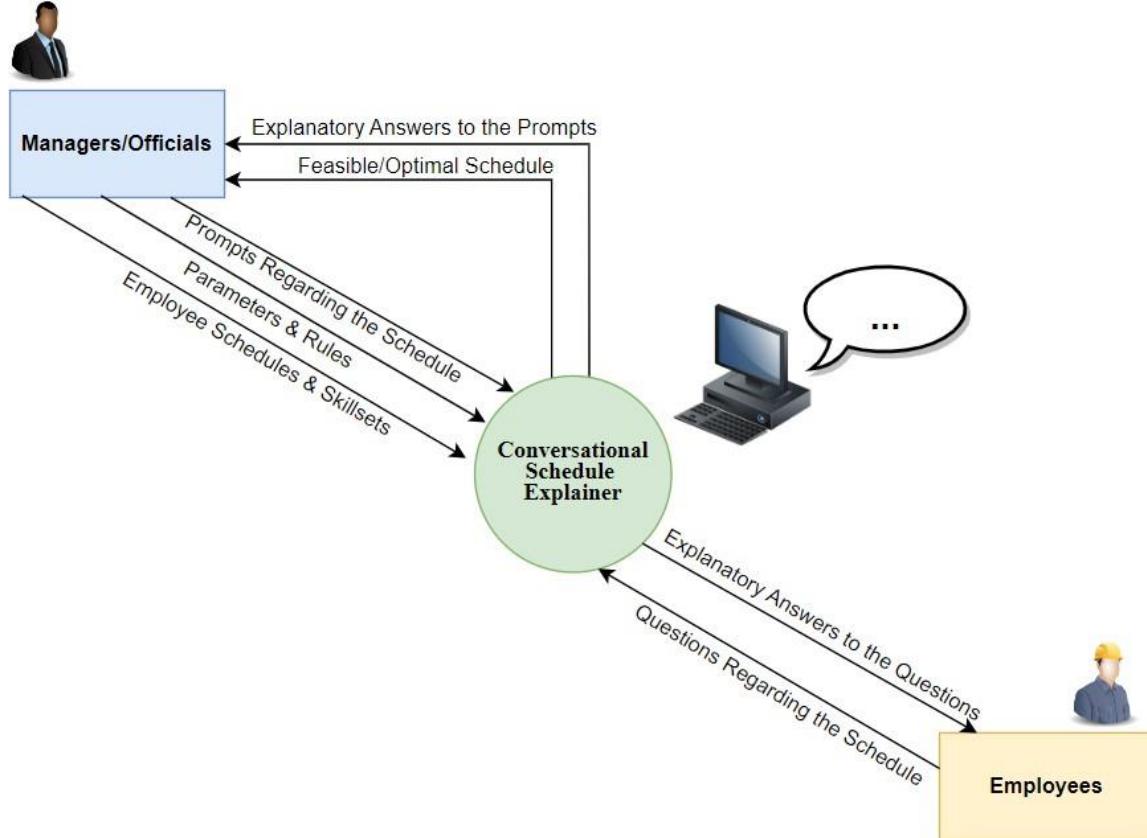


Figure 1. Context Diagram of the Model

3. Analysis for Solution/Design Methodology

3.1. Literature Review

The project is grounded in comprehensive literature review covering workforce scheduling, optimization models in service industries, and the application of AI for explainability. The literature indicates a significant gap in systems that not only optimize schedules but also provide clarity and reasoning behind scheduling decisions. Key references include studies on optimization algorithms, employee scheduling, and advancements in natural language processing for user interfaces.

The initial literature overview was on the schedule generation. This is performed in order to generate a schedule, as it is necessary to have one for explaining. Two papers are utilized majorly for that purpose.

“Personnel scheduling: A literature review” by Bergh et al. (2013): Their comprehensive review of personnel scheduling literature provided a broad understanding of various problem settings and technical features in the field. The classification methods and identification of trends in personnel staffing and scheduling research helped us in positioning our project within the current academic landscape and in identifying potential areas for innovative application of scheduling models.

“Employee Scheduling in Service Industries with Flexible Employee Availability and Demand” by Ağrahı et al. (2016): This paper, which considers employee scheduling in service industries with flexible availability and demand, was also useful for our project. Their focus on satisfying a wide range of constraints, including government and labor union regulations, while ensuring fairness and efficiency. Although we did not implement such constraints in our schedule, we have been inspired by the implementation of mixed - integer programming model used for scheduling in service industries.

After generating the schedule, we performed a literature review on explainable AI (XAI) implementations in scheduling problems, along with Natural Language Processing integration methods. Although there is some literature on both XAI and NLP integration to that in general, we observed a significant lack of literature regarding the integration of the emerging large language models such as ChatGPT for natural language processing.

"Natural Language Processing for Explainable Satellite Scheduling" by Powell et al. (2023), provided the concept of using NLP for making complex scheduling decisions understandable. This study's approach to autonomous decision-making and generating

explanations was particularly influential in how we developed our system to respond to 'why', 'what', 'when', and 'how' type queries.

"AI-assisted Schedule Explainer for Nurse Rostering" by Kristijonas Čyras et al. (2020) provided insights into creating a system that offers explanations for scheduling decisions, particularly in nurse rostering. Their work on creating an interactive and user-friendly interface, coupled with the delivery of actionable textual explanations, guided our approach to making our scheduling system more accessible and transparent to users.

3.2. Alternative Solution/Design Approaches

In developing our employee scheduling system, we navigated through different design approaches before decisively selecting a hybrid model. This decision was rooted in a commitment to not only achieve operational efficiency but also to ensure user engagement and clarity. Initially, an algorithmic solution was contemplated, focusing predominantly on optimizing schedules, with no explanations (i.e generating schedules based on a model without the XAI component). While this approach is successful in building a schedule, it lacks user interaction, a critical component for ensuring employee understanding and engagement which eventually leads to increased productivity and job satisfaction. Hence, we developed a hybrid model, which includes an integration of XAI components (underlying explainable AI functions, ChatGPT API, Streamlit user interface)

The hybrid model we chose integrates an obtained feasible or optimal schedule and underlying explanatory functions with an intuitive and interactive user interface, enhanced by ChatGPT's natural language processing capabilities and Streamlit's user-friendly interface. The core of this system is the underlying XAI functions that adeptly checks (and re-solves optimally when necessary) the scheduling constraints such as employee skills and shift timings, while also limiting the modifications in the schedule as well as the changes in the

objective function. Complementing this, the user interface, built with Streamlit, offers an accessible and engaging platform for employees to interact with the scheduling system. The integration of ChatGPT is pivotal here, as it maps the employee questions to the underlying explainability functions to user queries in natural language, making the system approachable and easy to use for all employees, regardless of their technical background.

This hybrid approach ensures that the shift assignments are not only feasible or optimal schedules, but also they are responsive and adaptable to user questions, providing a high level of clarity. It is this dynamic cooperation between a robust optimization model and an AI-driven, user-friendly interface that makes our system not just efficient but also inclusive and transparent. This thoughtful integration of optimization algorithms and AI-driven interaction exemplifies our commitment to enhancing operational efficiency while simultaneously elevating employee satisfaction in the scheduling process.

3.3. Assumptions

The system's design is based on specific assumptions. First of all, our model assumes there is an initial schedule adhering to constraints like skill requirements, shift timings, and legal working hours. If it is not built yet, the corresponding schedule is generated (in our case using Gurobi), adhering to these particular constraints. Secondly, since employee queries are diverse, ranging from simple information requests to complex explanations about scheduling decisions, we have to group the type of questions that employees will ask in general. Therefore the model is constructed assuming employees will ask most frequent questions, which our model can answer based on constraint violations, objective function changes, and re-solving the optimization model (with Gurobi) after fixing some variables and relaxing some constraints. In our problems, we assumed the primary objective of the schedule be to minimize the number of uncovered tasks, assuming several constraints:

- Employees are assigned tasks only if his/her assigned shift covers the task's duration.
- Employees must have at least two days off per week.
- After finishing a shift, an employee must have at least 13-hours rest before the next shift.
- Employees must possess the required skill for being assigned to a task
- An employee can do one task at a time (overlapping tasks can't be performed by one).

3.4. Brief Overview of the Selected Approach

The selected approach employs a combination of optimization models for scheduling, underlying explainability functions used for handling the questions, and ChatGPT for natural language processing. The system includes functions for answering FAQs, including query type questions, reasoning questions, and feasibility checks of schedule-related questions. The ChatGPT integration serves as an ergonomic user interface, interpreting queries in natural language and mapping them to the relevant function calls. Below in **Figure 2.** an instance of how our model works is shown.

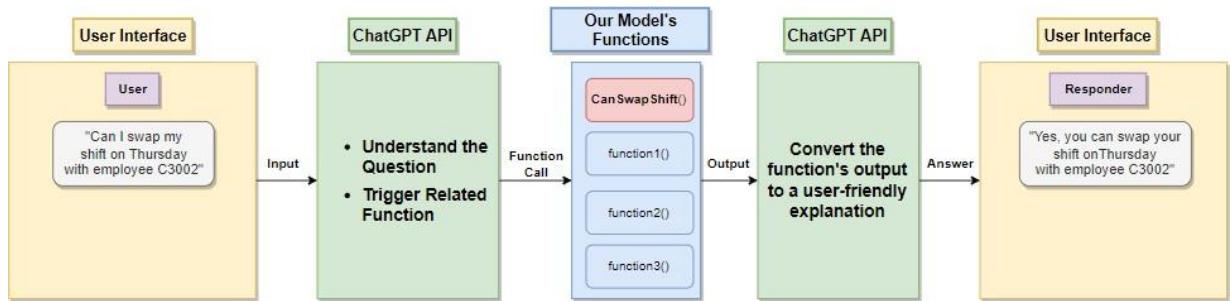


Figure 2. Exemplary View of How the Model Works

3.5. IE Tools & Methods Integration

In our project, the integration of various Industrial Engineering tools and techniques has been pivotal in developing a comprehensive employee scheduling system. At the start of the project, we used several mathematical modeling and integer programming methods. To tackle the complexities of the scheduling optimization problem, we utilized Gurobi, a powerful tool known for its efficiency in formulating and solving such problems.

Afterwards, the main body of the project, implementing the explanatory functions and then integrating them with ChatGPT is performed. This required us to first implement the search algorithms utilizing an advanced level of programming. This was essential to explain the scheduling decisions. Afterwards, we used Streamlit interface and ChatGPT to provide an ergonomic user experience. The role of ChatGPT, leveraging natural language processing, has been instrumental in our project. It serves as the bridge between the technical aspects of the scheduling system and the end-users, interpreting user queries and effectively communicating responses in a manner that is easily understandable.

To sum up, throughout the project a variety of industrial engineering approaches are utilized, including - but not limited to - Scheduling, Mathematical Modeling & Optimization, Integer Programming, Natural Language Processing, Programming, and Ergonomics along with tools such as Python and Gurobi.

4. Development of Solutions & Comparing Alternatives and Recommendation

The development of our employee scheduling system had a complex design process, focusing on creating a robust, transparent, and interactive solution for handling various frequent employee queries. Therefore, this approach was further reviewed through a comprehensive comparison with potential alternatives thus leading to a well-justified recommendation.

4.1 Detailed Technical Description of the Design Process

To effectively implement our project focusing on schedule explainability, it was essential to first create a schedule. We chose airport staff scheduling as our case study, characterized by continuous and fluctuating demands, necessitating flexible shift schedules. It's important to note, however, that this methodology is adaptable to various contexts, such as nurse scheduling.

Sets:

- T : Tasks
- E : Employees
- S : Shift start times

MATHEMATICAL MODEL

Parameters:

- t_{start}^t : Start of task t
- t_{end}^t : End of task t
- skill^t : Required skill for task t
- skillset_e : Skills of employee e
- shiftEnd_s : End of shift at time s

Decision Vars:

- $x_{e,s}$: 1 if employee e starts at s , else 0
- $y_{e,t}$: 1 if employee e covers task t , else 0
- z_t : 1 if task t is uncovered, else 0

Objective:

$$\text{Minimize } \sum_{t \in T} z_t$$

Constraints:

1. $\sum_{e \in E} y_{e,t} + z_t = 1, \quad \forall t \in T$
2. $y_{e,t} \leq \sum_{s=t_{\text{start}}}^{\text{shiftEnd}_t} x_{e,s}, \quad \forall e \in E, t \in T$
3. $\sum_{s \in S} x_{e,s} \leq 5, \quad \forall e \in E$
4. $x_{e,s'} + x_{e,s''} \leq 1 \text{ where } s' > s \text{ and } s' - s \leq 24, \quad \forall e \in E$
5. $y_{e,t} \leq 1 \text{ if } \text{skill}^t \in \text{skillset}_e, \text{ else } 0, \forall e \in E, t \in T$
6. $y_{e,t_1} + y_{e,t_2} \leq 1 \text{ for overlapping } t_1, t_2, \forall e \in E$

Figure 3. Mathematical Model of the Optimization Process

The mathematical model for the schedule is illustrated in **Figure 3**. As our project primarily concentrates on explaining an existing schedule rather than developing an optimized one, an in-depth exploration of the scheduling model is not the focus of this report.

The schedule is structured to allocate various tasks to a group of employees, where each task has a specified start and end time and requires certain skills. The employees, possessing various skills, are assigned to different shifts. A brief example of the schedule is presented for illustration in **Table 1**.

Task ID	Required Skill	Task Start Time	Task End Time	Assigned Employee	Employee Skillset	Shift Index	Shift Start Time	Shift End Time
1	7	2023-07-10 15:15:00	2023-07-10 23:45:00	C0161	[3, 4, 5, 7]	11	2023-07-10 15:00:00	2023-07-11 02:00:00
10	6	2023-07-10 15:30:00	2023-07-10 19:00:00	C0129	[3, 4, 5, 6]	11	2023-07-10 15:00:00	2023-07-11 02:00:00
13	6	2023-07-11 02:00:00	2023-07-11 03:20:00	C0307	[3, 4, 5, 6]	22	2023-07-11 02:00:00	2023-07-11 13:00:00
14	6	2023-07-11 04:00:00	2023-07-11 05:00:00	C0307	[3, 4, 5, 6]	22	2023-07-11 02:00:00	2023-07-11 13:00:00
15	6	2023-07-11 04:15:00	2023-07-11 05:00:00	C0359	[3, 6, 7]	24	2023-07-11 04:00:00	2023-07-11 13:00:00

Table 1. Snapshot from the Schedule

The primary objective of the schedule is to minimize the number of uncovered tasks, adhering to several constraints:

- Workers are assigned tasks only within their assigned shift time intervals.
- Workers are entitled to at least two days off per week for rest.
- A minimum of 13-hour rest is mandatory after completing a shift.
- Tasks can only be assigned to employees who possess the task's required skill.
- Tasks with overlapping time intervals can not be handled by the same employee.

The schedule is thus a solution aimed at minimizing uncovered tasks while respecting these constraints. These constraints, which form our assumptions, can be modified according

to different operational requirements and play a crucial role in the explanation process for query feasibility.

Using Python and Gurobi, we generated a schedule, adopting certain task groupings and sequencings to reduce computational time, as the complexity of the problem (NP-hard) escalates with increased data size.

Although the focus of this project is not on the generation but rather the explanation of the schedule, understanding the objective function and constraints is vital. They are especially instrumental in addressing request-type questions. For instance, when a request violates a constraint, that becomes the basis for its rejection. The objective function is crucial for evaluating the company's loss due to a particular request. For example, a feasible request might be denied if it leads to an unacceptable decrease in covered tasks. The nuances of these processes will be elaborated upon in subsequent chapters. Below in **Figure 4**. The general algorithmic logic behind our model is shown in detail.

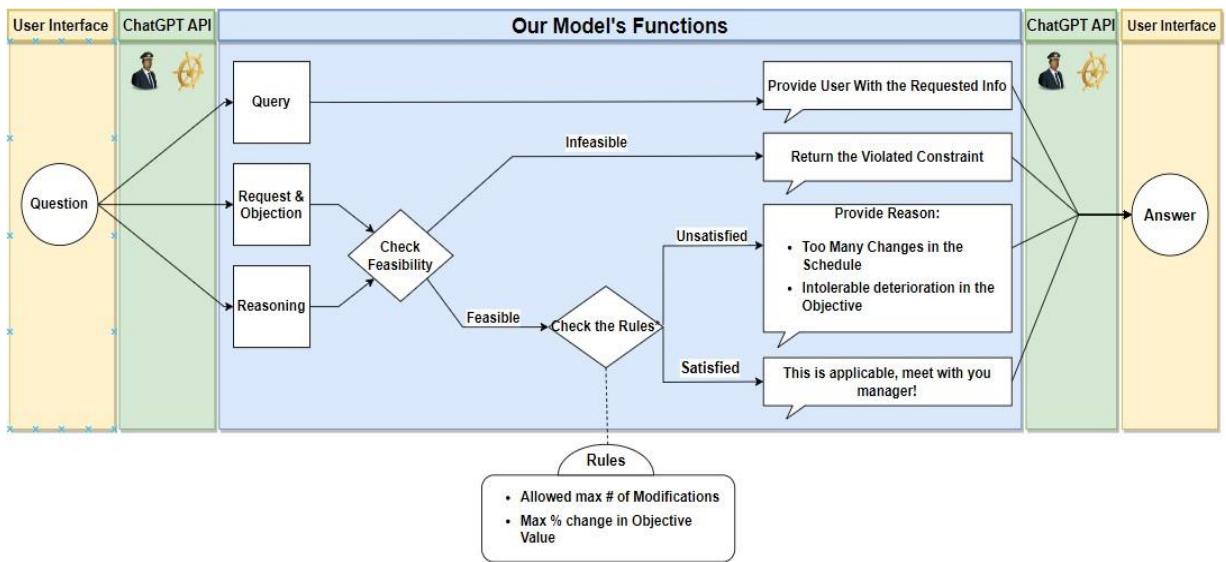


Figure 4. Overview of the Model

4.1.1 List Of Frequently Asked Questions

To begin with, we made a list of frequently asked questions and requests coming from employees which will serve as input scenarios we examine. Fundamentally, we split them in three main groups:

- ***Query***: Questions for information retrieval (ex: Who works with me tomorrow?)
- ***Request***: Questions involving demands or objections (ex: Can I swap my shift on [day1] with Ali's on [day2]?)
- ***Reasoning***: Questions asking for underlying logic (why – questions). (ex: Why am I not assigned to task [t]?)

These groups intuitively make sense, but that is not the only reason for grouping them this way. Each group requires different algorithms for answering.

The query type questions are the easiest ones, as they can directly be taken from the information by simply retrieving the existing information. Though they are simple to answer, one should note that these types of questions are asked frequently.

The request questions mostly check for the feasibility or applicability of the request by using certain search algorithms. For instance, in the example we mentioned, the request for switching tasks between employees, we analyze the schedule's constraints after the proposed change. If any of the constraints is violated, the request is infeasible due to that constraint. Then, this result is given to the employee as an answer.

Reasoning questions differ from the other two types in one very important aspect. They frequently require an optimization model, or solving the existing schedule with a proposed optimization model after fixing some variables according to the request.

Based on these groupings, we have generated a list of frequently answered questions in **Table 2**. which will be answered via backend functions.

QUERY	"Who is on the same shift as me on [date]?" "Who else has the same skill set as me and is working on [shift time]?" "What is my weekly shift schedule?" "Which tasks are assigned to me on [Date]?"
REQUEST	"Can I swap my shift on [date] with [Employee's Name]?" "I can't come to my shift on [Date]. Can it be rescheduled?" "Can I perform task [t] instead of task [k]?" "Is it possible to start my shift on [Date] [n] hours later?" "Can I avoid being scheduled with [Employee's Name] on [Date]?"
REASONING	"Why am I not assigned to shift [ID]?" "Why am I not assigned to Task [ID]?"

Table 2. Frequently Asked Questions

Below, each type of question (query, request, reasoning) and the algorithms for answering them will be provided in detail. Python implementations of the functions are in Appendix.

4.1.2 Answering Query Questions

Firstly, we aimed to answer basic query questions that employees would ask. Using the FAQ that we have obtained, we created functions to answer query prompts, which do not need any additional optimization and can be answered by basic comparisons and information retrieval. The query type functions, therefore, are the fastest working functions of our model. Below, the query type of our functions are explained in detail with required parameters and working conditions, along with provided questions.

Function 1 - *find_employees_on_same_shift(employeeID, date)*

A possible query for this function would be: "*Who is on the same shift as me on 4 May 2023, I am employee C0007.*" The *find_employees_on_same_shift* function is used to

find employees working on the same shift for a specific employee on a specified date. The function takes the ID of the questioning employee, specified date. After this, it checks if the questioning employee has a shift on the specified date. Then it returns the list of other employees on the specified date.

Function 2 - *find_employees_with_same_skillset*(employeeID, shift_time)

A possible query for this function would be: "***Who else has the same skill set as me and is working between 16.00 - 03.00?***" The *find_employees_with_same_skillset* function is used to find employees who have the same skill set as the questioning employee on a specified shift. Function takes questioning employee's ID, specified shift time, and schedule data as input. Then it returns the list of employees who have the same skill set with the questioning employee on the specified shift.

Function 3 - *get_shift_schedule*(employeeID, reference_date)

A possible query for this function would be: "***Can you give me my weekly schedule on date 11 June 2024?***" The *get_shift_schedule* function is used to get schedule information for the questioning employee. It takes an employee's ID, schedule data, and a reference date as input. Then it returns questioning employees 1-week schedule as the output on the specified date.

An example answer generation process of a query function is shown in **Figure 5**. And **Figure 6**.

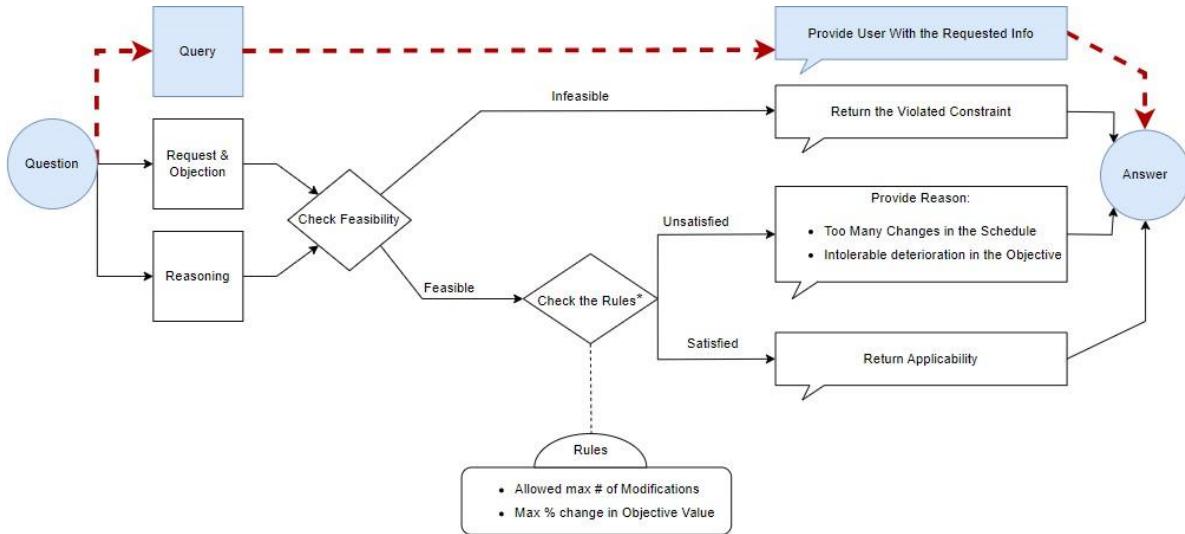


Figure 5. Example Answer Generation of a Query Function

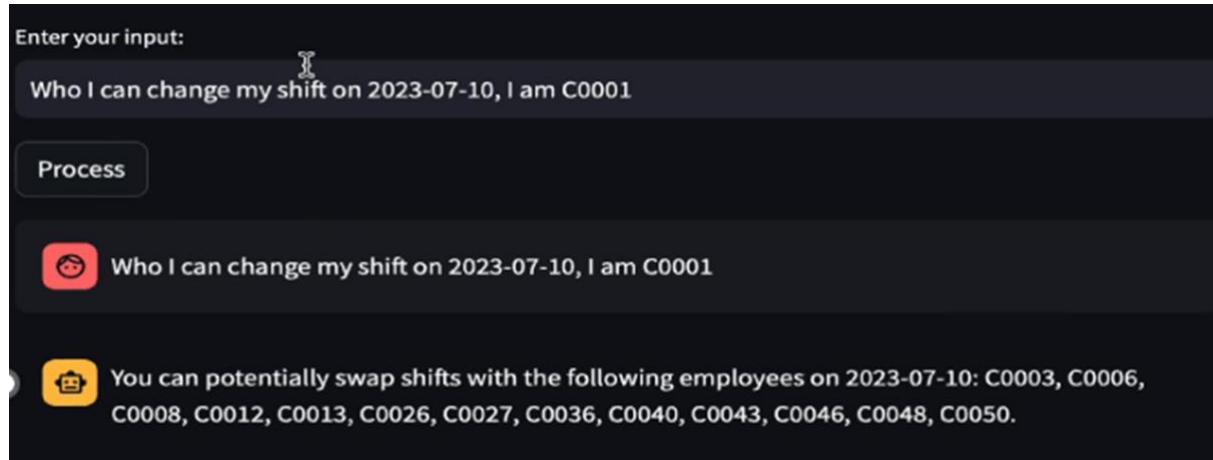


Figure 6. Screenshot of the Interface Answering a Query Question

4.1.3 Answering Request Questions

Function 4 - *can_swap_tasks*(*schedule*, *employee1_ID*, *employee2_ID*, *employee1_task*, *employee2_task*)

A possible query for this function would be: "***Can I perform task 11 instead of task 17?***" The *can_swap_tasks* function is used for checking if an employee can switch a specific task with another specific employee's specific task. It basically checks if this

particular switch is possible by checking the constraints. Specifically, these constraints include skill level constraints, shift time constraints and overlapping constraints. According to the result of this check, the function returns an answer.

Function 5 - *can_swap_shifts*(employee1_ID, employee2_ID, employee1_date, employee2_date)

A possible query for this function would be: "***Can I swap my shift on 28 September 2014 with Murat Tutar?***" The function *can_swap_shifts* is used for checking if an employee can switch a specific shift on a specific date with another employee. It checks constraints such as the minimum rest time, 5-shift per week limit and skillset match. According to the result of this check, the function returns an answer.

Function 6 - *find_possible_swaps*(employeeID, shift_date)

A possible query for this function would be: "***Whom can I change my shift on 7 June 2023 with? I am employee C0007.***" The function *find_possible_swaps* identifies potential shift-swap candidates for a specified employee on a given date, considering factors like skills and shift limits. It takes the employee code (*employeeID*) and shift date (*shift_date*) as inputs, utilizing the *can_swap_shifts* function for feasibility checks. The output is a list of employees (*possible_swaps*) eligible for shift swaps. If no candidates are found, a corresponding message is returned. This function aids in autonomously proposing feasible alternatives for shift adjustments in the scheduling process.

An example answer generation of request functions is shown in **Figure 7.** and **Figure 8.**

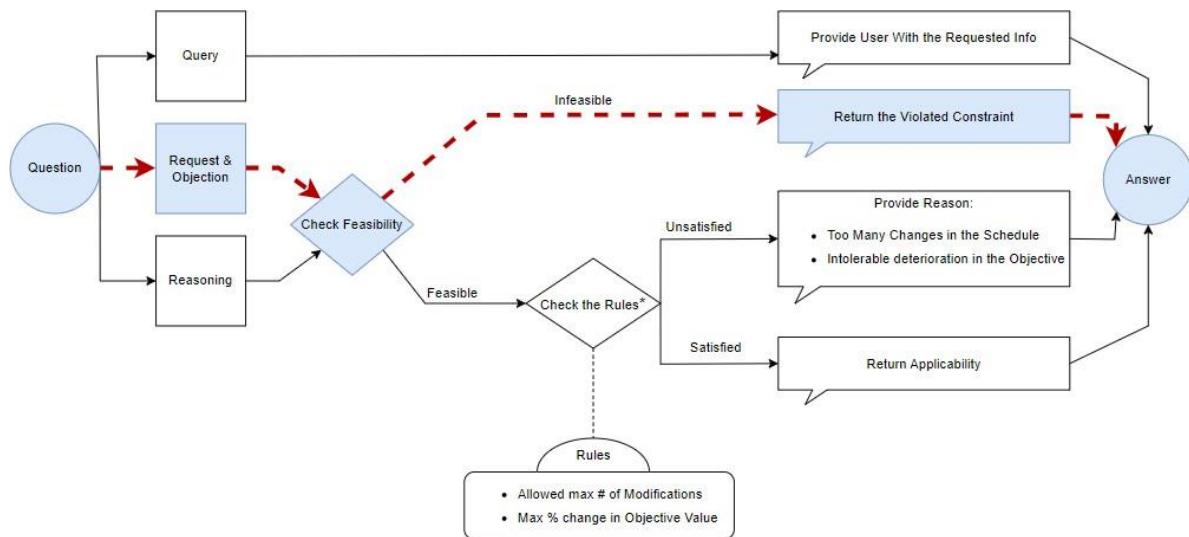


Figure 7. Example Answer Generation of a Request Function

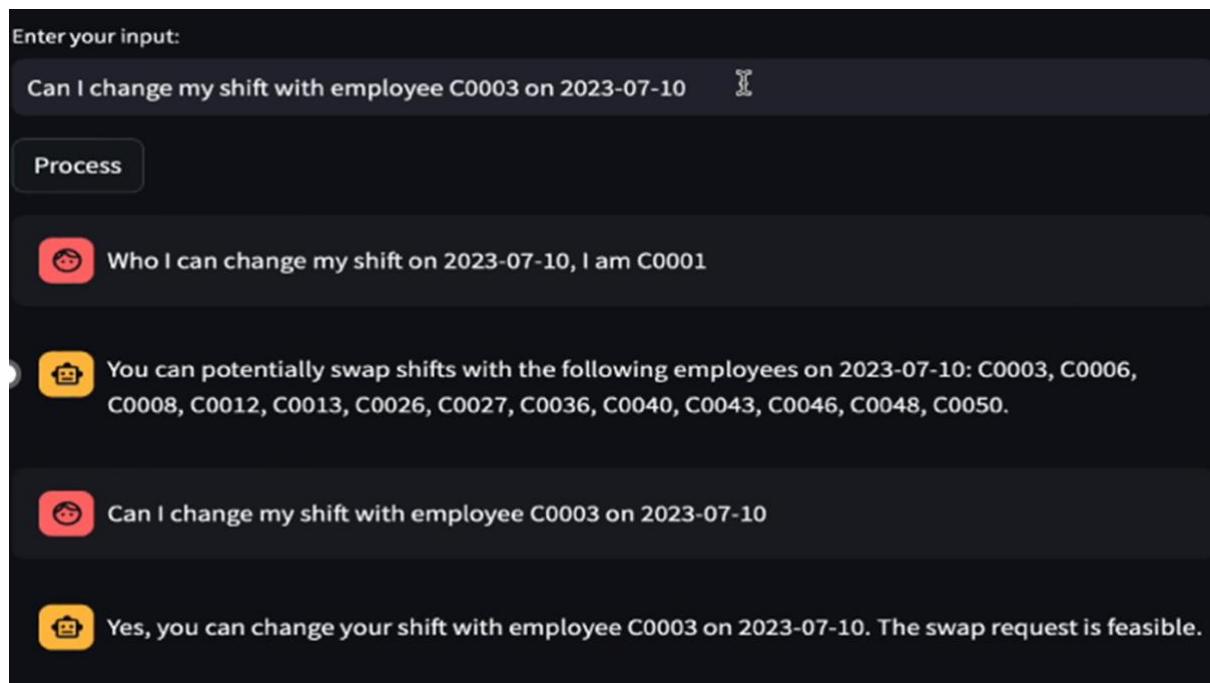


Figure 8. Screenshot of the Interface Answering a Request Question

4.1.4 Answering Reasoning Questions

In addressing reasoning questions within our project, a distinct approach was necessary, because they may often involve re-optimizing the existing schedule with potential adjustments (i.e fixing some decision variables or relaxing constraints of the original model). Hence, these reasoning questions can be analyzed in three different scenarios:

- I. **Constraint Violation Scenario:** If an employee's request for a specific task or shift assignment violates a constraint, it's infeasible. This situation is similar with the request-type questions, where the primary reason for rejection is the violation of preset constraints.
- II. **Feasibility with Objective Worsening:** In cases where the assignment is feasible but leads to a deterioration of the objective (e.g., an increase in the number of uncovered tasks), the system needs to assess the extent of this impact. If the negative impact exceeds a predefined tolerable limit, the assignment is rejected, explaining to the employee the financial or operational repercussions of such a change.
- III. **Multiple Optimal Schedules Scenario:** Certain instances reveal multiple optimal scheduling solutions. In such cases, the employee's assignment to the queried task or shift might be feasible without violating constraints or worsening the objective. However, implementing this change could lead to significant alterations in the overall schedule. The decision here hinges on predefined rules, such as limiting the total number of task reassessments in the schedule. If the proposed modified schedule induces more changes than allowed, it's rejected to maintain schedule stability. Conversely, if the change is within acceptable limits, it can be approved.

It's crucial to recognize the computational complexity inherent in reasoning questions, especially when they necessitate optimization. As the problem size grows, these queries

become increasingly challenging to answer in a reasonable timeframe, highlighting the NP-hard nature of the task.

In the frequently asked questions list, two reasoning questions are specifically addressed. For each, we developed functions to generate explanations. These functions operate as follows:

Function 7 - *why_not_assigned_to_shift*(employee_code, shift_id)

A possible question for this function would be: "***Why am I not assigned to shift 15?***". The function takes the employee's name, the queried shift as inputs. It then determines if the request is infeasible due to constraint violations. If feasible but detrimental to the objective, the current version of the system does not yet provide a detailed explanation to the employee. However, if the change is feasible and doesn't worsen the objective, the system evaluates the necessary alterations. A significant change in the schedule or an unacceptable decline in the objective value leads to rejection, with explanations provided to the employee.

Function 8 - *why_not_assigned_to_task*(employee_code, task_id)

A possible question for this function would be: "***Why am I not assigned to task 81?***" This function operates similarly, with the task being the focus instead of the shift. It assesses the feasibility, impact on the objective, and the extent of schedule alterations required for accommodating the employee's request.

Function 9 - *reschedule_shift*(employee_code, shift_id, shift_id_wanted)

A possible question for this function would be: "***I can't come to my shift on 17 December 2023. Can it be rescheduled to 24 December 2023?***" Again, the logic is very similar. We fix the current shift assignment's decision variable to zero, and the requested

date's assignment to 1. Then, if feasible, we re-solve the optimization model. Additionally, the same function can be used for the following question too: "*Is it possible to start my shift on 13 October 2024 7 hours later?*" Similar to the previous FAQ, if feasible, this time the current shift's decision variable is set to zero and the requested shift assignment to 1, and then the Gurobi re-solves the problem.

Through these processes, the system aims to provide clear and reasoned explanations to employees, enhancing their understanding of scheduling decisions and fostering transparency in the workplace.

An example answer generation of reasoning functions is shown in **Figure 9.** and **Figure 10.**

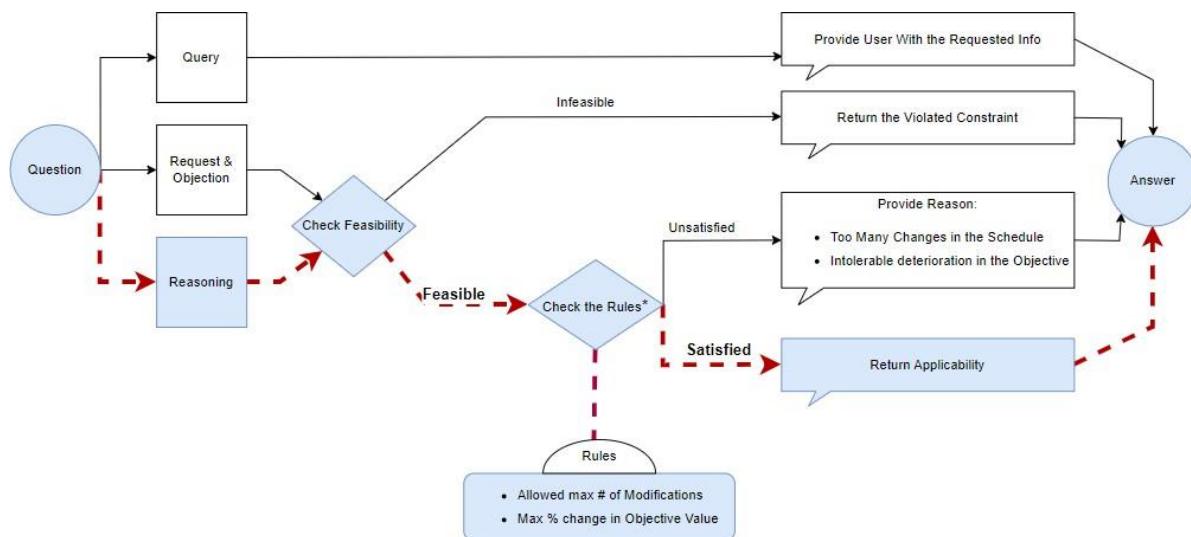


Figure 9. Example Answer Generation of a Reasoning Function

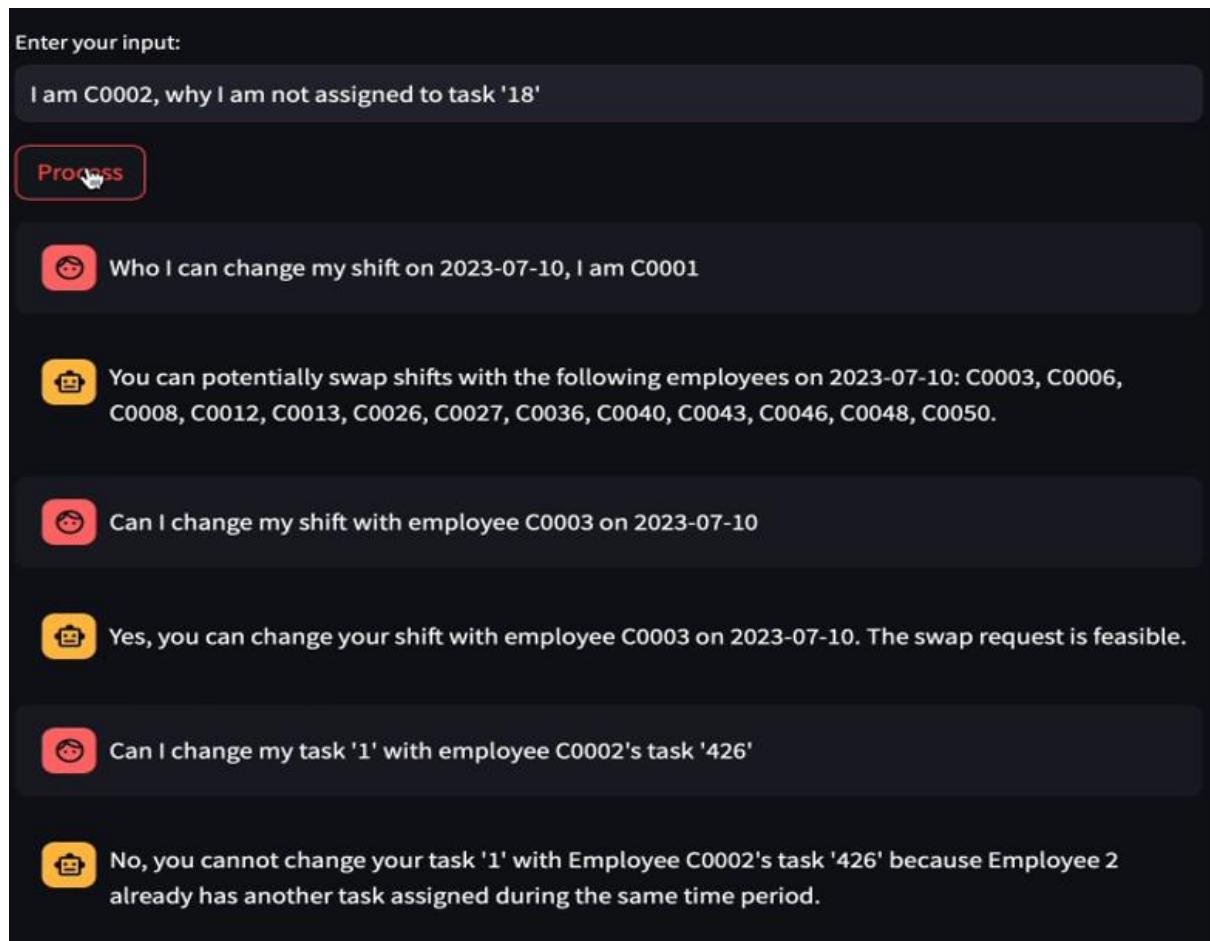


Figure 10. Screenshot of the Interface Answering a Reasoning Question

4.2 Creation of User Interface

An essential first step toward guaranteeing the system's usability and accessibility was the development of the user interface. Using the open-source Python library Streamlit, we created an interface that prioritizes ergonomic functionality and user experience. This interface offers a smooth question entry process and is the first point of contact for stakeholders. This integration not only provides better visibility but also takes into account the various technical backgrounds of users, guaranteeing an inclusive and user-friendly interface. The user interface plays a major role in the overall success of the project by

emphasizing clarity and ease of navigation, which creates an environment that facilitates productive and positive interactions between users and the system.

4.3 Integration of ChatGPT

The integration of ChatGPT API serves as a fundamental component in enhancing the system's natural language processing ability and a proficient dialogue between the user and system. With ChatGPT serving as an NLP tool, this process involves interpreting user input, identifying relevant underlying functions and related parameters, and then calling the relevant Python functions.

The system's recreation of chatbot functionality is noteworthy since it allows users to view previously sent messages and creates a personalized, contextually aware interface. In addition, ChatGPT API is essential for converting complex Python outputs into understandable messages so that users with different levels of technical proficiency are able to acquire the information. Therefore, the integration of ChatGPT API is a fundamental feature that enhances the project's overall usability and inclusivity, as it not only enables efficient communication but also makes accessible system access.

4.4 Comparison and Evaluation

The evaluation process focused on the accuracy of the responses, the efficiency of the system in handling different types of queries, and user satisfaction. As a result, the hybrid approach combining the mathematical model's constraints and objectives, generated schedule, underlying explanatory functions, and integration with an NLP tool (ChatGPT) was recommended. This approach was justified by its ability to handle a wide range of queries effectively while maintaining the integrity of the schedule. The resulting solution met the

project's core requirements of providing clear, reasoned explanations for scheduling decisions and efficiently handling schedule-related queries.

On one hand, in terms of implementation feasibility, the solution's design considered practical implementation aspects, ensuring that it could be integrated into existing operational frameworks with relative ease. On the other hand, the solution demonstrated a high degree of robustness, effectively handling diverse queries ranging from simple information requests to complex scheduling explanations. On the sustainability and data management aspect, however, the system requires regular updates to the scheduling model and continuous learning for the AI component to stay relevant and effective.

5. Suggestions for Successful Implementation

The design is implemented in phases and in a methodical manner. The project starts with creating an easy-to-use user interface with the Streamlit framework, which guarantees a user-friendly interface for interaction. This interface offers an ergonomic platform that improves accessibility for users with different levels of technical proficiency in addition to acting as a point of entry for user inquiries.

The user interface, ChatGPT API, and the underlying Python functions are all integrated into the design, which merges seamlessly with the overall architecture. User queries are routed through the Streamlit-powered user interface, which provides a friendly front end for complex backend operations. ChatGPT's natural language processing powers are essential for interpreting and processing user inputs and establishing a smooth communication channel with the underlying Python functions. These operations, which make up the system's fundamental logic, involve producing relevant answers and justifications to

the questions asked by stakeholders. A thorough and effective user experience is achieved by the integration's seamless transition from user interaction to natural language processing to the complexities of backend functionality.

Making sure the interface is user-friendly and adaptable to different levels of technical proficiency by giving priority to user-centric design principles and to improve the user experience, carrying out iterative feedback loops and usability testing is fundamental. After implementation, keeping an eye on user interactions, testing the system thoroughly, and establishing a solid integration between the components is also a must.

System architecture needs to be flexible for future upgrades and scalability. Planning ahead for scalability needs and use modular design to ensure smooth integrations necessary for future success of implementation. A proactive maintenance program is needed that includes frequent bug fixes and updates to guarantee ongoing relevance and dependability.

Periodic revisions are necessary for optimal system performance in order to accommodate changing user needs and technological advancements. To keep ChatGPT's natural language processing capabilities up to date with changing linguistic patterns and user inquiries, it should undergo frequent evaluations and updates. Furthermore, the underlying Python functions ought to be updated in light of the model's changing goals and limitations (i.e objective and constraints). Periodic improvements to the user interface can help it become more usable and incorporate user feedback. Periodic systemic reviews, timed to correspond with the rate of technological development, are an important factor in maintaining the longevity and applicability of the developed explanation generation system.

6. Conclusions and Discussion

The integration of industrial engineering tools, techniques, and methods was essential for the success of this project. A wide range of methods including scheduling, mathematical modeling, optimization, and mixed-integer programming are used for generating the schedule and answering the reasoning questions that require optimization. Search algorithms that require advanced programming skills are used for answering the request questions that require feasibility checks. Also, natural language processing is used for understanding the questions and detecting relevant parameters. Last but not least, the usage of ChatGPT and Streamlit provided an ergonomically superior user experience.

The design's dedication to transparency, ease of comprehension, and user-centricity makes it valuable and significant. The system converts a black-box schedule into an understandable white-box, promotes improved comprehension of schedule decisions and satisfaction among individuals working there due to an increased level of clarity.

From an economic perspective, the project's significance in workforce management is positioned by its handling of a great variety of questions and its seamless integration with ChatGPT API, making it a unique and promising solution for the industry that saves time and money by increasing productivity thanks to an increased level of employee job satisfaction . Also, the potential to streamline administrative processes and enhance operational efficiency contributes directly to cost savings for organizations.

Environmentally, the optimized scheduling protocols hold the promise of reducing resource usage, thereby making a positive contribution to minimizing the carbon footprint of companies.

The project's implementation also must take ethical considerations with a careful approach. The model must be trained to identify which data can be shared with employees in order to protect sensitive information, such as health-related constraints. Similarly, the improper use of query questions, which might reveal private information about other employees, are risky. Hence, in order to lay the groundwork for ethical data handling, it is vital that explicit consent be obtained before sharing an employee's schedule information. Strong cybersecurity measures are required to protect employee privacy due to the growing threat of hacking, particularly in settings like airport operations. Above all, the system's answers must respect different legal requirements of each country or institution.

The design has a significant social impact and encourages a more satisfied and involved workforce. The project may lead to improvements in work-life balance and job satisfaction, which could have an impact on society as a whole. This project is essentially an example of how the combination of AI and industrial engineering can produce comprehensive, creative, and morally sound workforce management solutions. The project's long-term significance in the field of industrial engineering is established by the combined impact of its diverse contributions, which include advances in technology, economic efficiency, environmental sustainability, ethical considerations, and societal welfare.

7. References

1. Van den Bergh, J., Beliën, J., De Bruecker, P., Demeulemeester, E., & De Boeck, L. (2013). Personnel scheduling: A literature review. *European Journal of Operational Research*, 226(3), 367-385.
2. Ağralı, S., Taşkın, Z., & Ünal, A. T. (2016). Employee Scheduling in Service Industries with Flexible Employee Availability and Demand. *Omega*, 66.
3. Čyras, K., Karamlou, A., Lee, M., Letsios, D., Misener, R., & Toni, F. (2020). AI-assisted Schedule Explainer for Nurse Rostering. In Proc. of the 19th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2020), Auckland, New Zealand, May 9–13, 2020, IFAAMAS, 3 pages.
4. Powell, C., Berquand, A., & Riccardi, A. (2023). Natural Language Processing for Explainable Satellite Scheduling. In SPACEOPS 2023: The 17th International Conference on Space Operations, Dubai, United Arab Emirates, March 6-10, 2023, #349, 14 pages.

8. Appendix

Function A - *can_swap_tasks*

```
def can_swap_tasks(emp1_code, emp1_identifier:int, emp2_code,
emp2_identifier:int):

    emp1_identifier = int(emp1_identifier)
    emp2_identifier = int(emp2_identifier)

    # Find the tasks for both employees
    emp1_task = schedule_data[(schedule_data['AssignedEmployee'] ==
emp1_code) &
                               (schedule_data['Index'] == emp1_identifier)]

    emp2_task = schedule_data[(schedule_data['AssignedEmployee'] ==
emp2_code) &
                               (schedule_data['Index'] == emp2_identifier)]

    # Check if the tasks exist
    if emp1_task.empty or emp2_task.empty:
        return f"One or both of the specified tasks do not exist for given
employees."

    # Check if the skill levels match for the tasks
    if emp1_task['Skill'].iloc[0] != emp2_task['Skill'].iloc[0]:
        return f"The skill level required for the tasks does not match."

    # Check if the task times are within the employees' shift times
    if not (emp1_task['Task_Start'].iloc[0] >=
emp1_task['Shift_Start'].iloc[0] and
            emp1_task['Task_End'].iloc[0] <= emp1_task['Shift_End'].iloc[0]):
        return f"Employee 1's task time is not within the shift time."

    if not (emp2_task['Task_Start'].iloc[0] >=
emp2_task['Shift_Start'].iloc[0] and
            emp2_task['Task_End'].iloc[0] <= emp2_task['Shift_End'].iloc[0]):
        return f"Employee 2's task time is not within the shift time."
```

```

# Check for overlapping tasks with other tasks in the schedule for the
same employee

for other_task in schedule_data[(schedule_data['AssignedEmployee'] ==
emp1_code) & (schedule_data['Index'] != emp1_identifier)].itertuples():

    if (emp2_task['Task_Start'].iloc[0] < other_task.Task_End and
        emp2_task['Task_End'].iloc[0] > other_task.Task_Start):

        return f"Employee 1's task overlaps with another task assigned to
Employee 2."

    for other_task in schedule_data[(schedule_data['AssignedEmployee'] ==
emp2_code) & (schedule_data['Index'] != emp2_identifier)].itertuples():

        if (emp1_task['Task_Start'].iloc[0] < other_task.Task_End and
            emp1_task['Task_End'].iloc[0] > other_task.Task_Start):

            return f"Employee 2's task overlaps with another task assigned to
Employee 1."

# Assuming the rest of the checks pass

return f"The task swap is possible."

```

Function B - *can_swap_shifts*

```

def can_swap_shifts(emp1_code, emp1_date, emp2_code, emp2_date):

    # Convert string dates to datetime objects for comparison
    emp1_date = pd.to_datetime(emp1_date).date()
    emp2_date = pd.to_datetime(emp2_date).date()

    # Filter the schedule for the two employees on the given dates
    emp1_shifts = schedule_data[(schedule_data['AssignedEmployee'] ==
emp1_code) &
                               (schedule_data['Shift_Start'].dt.date ==
emp1_date) ]

    emp2_shifts = schedule_data[(schedule_data['AssignedEmployee'] ==
emp2_code) &
                               (schedule_data['Shift_Start'].dt.date ==
emp2_date) ]

    # Check if employees are working on specified dates
    if emp1_shifts.empty or emp2_shifts.empty:

```

```

        return f"One or both employees are not working on the specified
dates.""

    # Check if the employees have the same skillsets to be able to swap
    shifts

    emp1_skills = (skills_data.loc[skills_data['EmployeeCode'] == emp1_code,
'SkillCode'].values[0])

    emp2_skills = (skills_data.loc[skills_data['EmployeeCode'] == emp2_code,
'SkillCode'].values[0])

    if emp1_skills != emp2_skills:

        return f"The employees do not have matching skillsets."

    # Check the 5-shifts-per-week limit considering unique shifts

    emp1_weekly_shifts = schedule_data[(schedule_data['AssignedEmployee'] ==
emp1_code) &

(schedule_data['Shift_Start'].dt.isocalendar().week ==
emp1_shifts['Shift_Start'].dt.isocalendar().week.iloc[0])]

    emp2_weekly_shifts = schedule_data[(schedule_data['AssignedEmployee'] ==
emp2_code) &

(schedule_data['Shift_Start'].dt.isocalendar().week ==
emp2_shifts['Shift_Start'].dt.isocalendar().week.iloc[0])]

    # Removing duplicate shifts based on the start and end times

    emp1_unique_shifts =
emp1_weekly_shifts.drop_duplicates(subset=['Shift_Start', 'Shift_End'])

    emp2_unique_shifts =
emp2_weekly_shifts.drop_duplicates(subset=['Shift_Start', 'Shift_End'])

    if len(emp1_unique_shifts) > 5 or len(emp2_unique_shifts) > 5:

        return f"Shift swap would cause an employee to exceed the
5-shifts-per-week limit."

    # Check for 13-hour rest time between shifts

    # Find the shifts immediately before and after the swap dates for both
employees

    emp1_prev_shift = schedule_data[(schedule_data['AssignedEmployee'] ==
emp1_code) &

(schedule_data['Shift_End'] <
emp1_shifts['Shift_Start'].min())]

```

```

        emp1_next_shift = schedule_data[(schedule_data['AssignedEmployee'] ==
emp1_code) &
                                    (schedule_data['Shift_Start'] >
emp1_shifts['Shift_End'].max())]

        emp2_prev_shift = schedule_data[(schedule_data['AssignedEmployee'] ==
emp2_code) &
                                    (schedule_data['Shift_End'] <
emp2_shifts['Shift_Start'].min())]

        emp2_next_shift = schedule_data[(schedule_data['AssignedEmployee'] ==
emp2_code) &
                                    (schedule_data['Shift_Start'] >
emp2_shifts['Shift_End'].max())]

# Check if there's enough rest between the end of the last shift and the
start of the swapped shift

    if not emp1_prev_shift.empty and (emp2_shifts['Shift_Start'].min() -
emp1_prev_shift['Shift_End'].max()).total_seconds() / 3600 < 13:
        return f"Not enough rest time before the shift for Employee 1."

    if not emp1_next_shift.empty and (emp1_next_shift['Shift_Start'].min() -
emp2_shifts['Shift_End'].max()).total_seconds() / 3600 < 13:
        return f"Not enough rest time after the shift for Employee 1."

    if not emp2_prev_shift.empty and (emp1_shifts['Shift_Start'].min() -
emp2_prev_shift['Shift_End'].max()).total_seconds() / 3600 < 13:
        return f"Not enough rest time before the shift for Employee 2."

    if not emp2_next_shift.empty and (emp2_next_shift['Shift_Start'].min() -
emp1_shifts['Shift_End'].max()).total_seconds() / 3600 < 13:
        return f"Not enough rest time after the shift for Employee 2."

return f"Swap request is feasible"

```

Function C - *check_shift_compatibility*

```

def check_shift_compatibility(emp_code, other_emp_code):

    # Check for any common shifts between the two employees

    common_shifts = schedule_data[(schedule_data['AssignedEmployee'] ==
emp_code) |
```

```

        (schedule_data['AssignedEmployee'] ==
other_emp_code) ]

# Drop duplicates to get unique shifts only
common_shifts = common_shifts.drop_duplicates(subset=['Shift_Start',
'Shift_End'])

# Group by shift start and end times and count unique employees in those
shifts
shift_counts = common_shifts.groupby(['Shift_Start',
'Shift_End'])['AssignedEmployee'].nunique()

# Filter shifts where both employees are scheduled
shared_shifts = shift_counts[shift_counts >= 1]

if shared_shifts.empty:
    return f"No common shifts are scheduled with the specified employee."

# If there are shared shifts, check if both employees have tasks in those
shifts
for shift_start, shift_end in shared_shifts.index:
    emp_tasks = schedule_data[(schedule_data['AssignedEmployee'] ==
emp_code) &
                               (schedule_data['Task_Start'] >=
shift_start) &
                               (schedule_data['Task_End'] <= shift_end)]
    other_emp_tasks = schedule_data[(schedule_data['AssignedEmployee'] ==
other_emp_code) &
                               (schedule_data['Task_Start'] >=
shift_start) &
                               (schedule_data['Task_End'] <=
shift_end)]

    # Check if both employees have tasks in the shared shift
    if not emp_tasks.empty and not other_emp_tasks.empty:
        return f"You have to work together in this shift."

    return f"There are common shifts, but you do not have tasks assigned in
those shifts with the specified employee."

```

Function D - *find_possible_swaps*

```
def find_possible_swaps(empl_code, empl_date):  
    possible_swaps = []  
  
    # Iterate through all employees to find potential swaps for emp1  
    for emp2_code in schedule_data['AssignedEmployee'].unique():  
        if emp2_code == empl_code:  
            continue # Skip emp1  
  
        # Check if emp1 can swap shifts with emp2 on the specified date  
        result = can_swap_shifts(empl_code, empl_date, emp2_code, empl_date)  
        #print("results is", result)  
  
        if result == "Swap request is feasible":  
            possible_swaps.append(emp2_code)  
  
    if not possible_swaps:  
        return "No employees available for shift swaps with Employee {} on  
{}.".format(empl_code, empl_date)  
  
    return "Possible swaps for Employee {} on {} with the following  
employees: {}".format(empl_code, empl_date, possible_swaps)
```

Function E - *generate_function*

```
def generate_function(function_name):  
    if function_name in function_data_cache:  
        return function_data_cache[function_name]  
  
    func = custom_functions[function_name]  
    params = list(inspect.signature(func).parameters.keys())  
    function_data = {  
        "name": function_name,  
        "description": func.__doc__,
```

```

    "parameters": {
        "type": "object",
        "properties": {param: {"type": "string", "description": param} for param in params},
        "required": params
    }
}

function_data_cache[function_name] = function_data
return function_data

```

Function F - *get_completion*

```

def get_completion(query, past_messages):
    user_message = {"role": "user", "content": query}
    past_messages.append(user_message)
    while True:
        response = openai.ChatCompletion.create(
            model="gpt-3.5-turbo-0613",
            messages=past_messages,
            functions=[generate_function(name) for name in
list(custom_functions.keys())],
            temperature=0,
            function_call="auto",
        )
        #print("response is: ", response)
        message = response["choices"][0]["message"]
        past_messages.append(message)

        if message.get("function_call"):
            print("Message with function call:", message)
            function_name = message["function_call"]["name"]
            function_args = json.loads(message["function_call"]["arguments"])
            func = custom_functions[function_name]

```

```

        print(f"Function Name: {function_name}")
        print(f"Arguments: {function_args}")

        signature = inspect.signature(func)

        no_default_arguments = [param for param in
signature.parameters if signature.parameters[param].default ==
inspect.Parameter.empty]

        # To get a list of arguments which do not have default values

        entered_messages = [message['content'] for message in
past_messages if message['role'] == 'user'][1:]

        entered_messages = "".join(entered_messages)

        missing_args = [key for key in no_default_arguments if key
not in function_args or function_args[key] not in entered_messages]

        #for arg in missing_args:
        #user_input = "02/01/2023"
        #function_args[arg] = user_input

        #if function_name == 'find_possible_swaps':
        #function_response1 = func(**function_args)

        #return function_response1

        function_response = func(**function_args)
        function_message = {
            "role": "function",
            "name": function_name,
            "content": function_response,
        }
        past_messages.append(function_message)
        #return function_response

    else:
        return message["content"]

```

Function G - *why_not_assigned_to_task*

```

def why_not_assigned_to_task(employee_code, task_id:int):

    task_id = int(task_id)
    if task_id not in tasks:
        return f"Task ID does not exist."

    if employee_code not in employees:
        return f"Employee does not exist."

    task = tasks[task_id]
    employee = employees[employee_code]

    # Check skill requirement and working time
    if task.skill_required not in employee.skills:
        return "Infeasible: Employee lacks the required skill."
    if not any(shift.start_time <= task.start_time and shift.end_time >=
    task.end_time for shift in employee.shifts):
        return f"Infeasible: Employee is not working during the task period."

    # Check for task overlap
    for other_task_id in tasks:
        if other_task_id != task_id and x[employee_code, other_task_id].X >
0.5:
            other_task = tasks[other_task_id]
            if not (other_task.end_time <= task.start_time or task.end_time
<= other_task.start_time):
                return f"Infeasible: Task overlaps with task {other_task_id}."

    # Optimization with the new task assignment
    original_values_x = {var: var.X for var in x.values()}
    original_values_y = {var: var.X for var in y.values()}

        model.addConstr(x[employee_code,      task_id]      ==      1,
name=f"fix_{employee_code}_{task_id}")

    model.optimize()

```

```

if model.status != GRB.OPTIMAL:
    return f"Infeasible schedule."

# Check optimality and number of changes

    num_changes = sum(1 for var in x.values() if abs(var.X - original_values_x[var]) > 0.5)

    original_obj_val = sum(original_values_y[y[t]] for t in tasks)
    new_obj_val = sum(y[t].X for t in tasks)

    optimality_loss = (new_obj_val - original_obj_val) / max(1, original_obj_val)

    if optimality_loss > optimality_tolerance or num_changes > max_changes_allowed:
        # Revert changes and provide feedback
        for var, value in original_values_x.items():
            var.Start = value
        for var, value in original_values_y.items():
            var.Start = value

        change_info = ""
        if num_changes > max_changes_allowed:
            change_info = f"Not Applicable: Requires too many adjustments in schedule (causing at least {num_changes} changes of tasks/shifts/assignments)."
        if optimality_loss > optimality_tolerance:
            change_info += f"Not Applicable: Worsens schedule's optimality by more than {optimality_tolerance*100}%.\\nIn other words, the company loses too much"

    return change_info

return f"Assignment is feasible with minimal schedule disruption."

```