

# LanguageTeams Planning App

Realization document

Rob Verbeek  
Student Bachelor Applied Computer Science

# Table of Contents

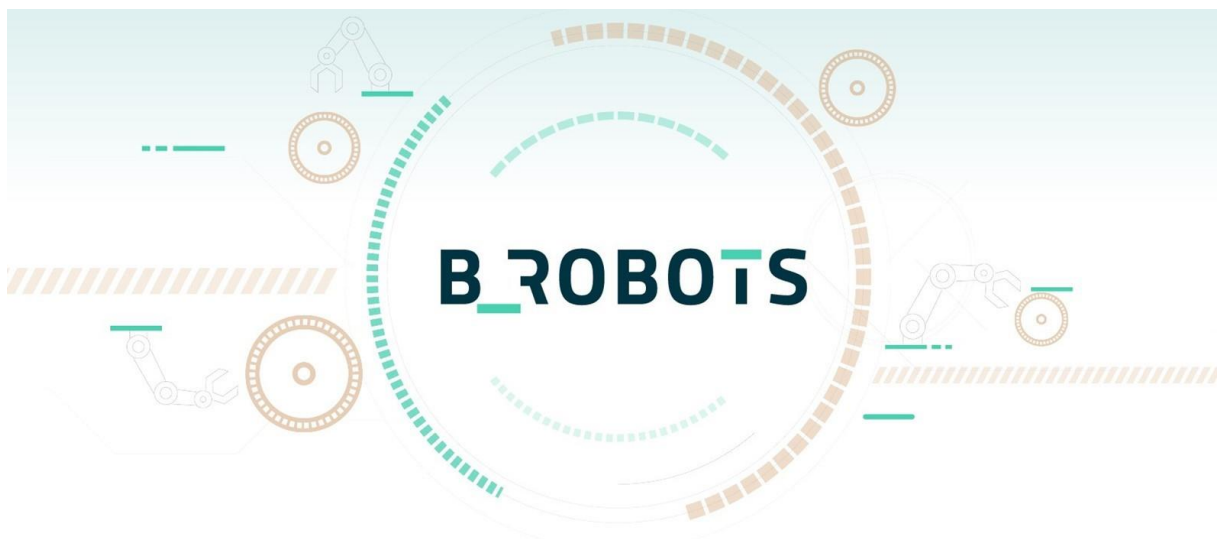
<b>1. INTRODUCTION</b>	<b>3</b>
<b>2. ANALYSIS</b>	<b>4</b>
2.1. Project Scope	4
2.2. Application	5
2.2.1. Django Web Framework	5
2.2.2. Azure Deployment	5
2.2.3. Docker in Development	5
2.2.4. Frontend Styling: CSS and Bootstrap	6
2.3. Genetic Algorithm Planner	7
2.4. Agentic AI Planner	8
2.5. Conclusion	9
<b>3. REALISATION</b>	<b>10</b>
3.1. Data Preparation	10
3.2. Genetic Algorithm Planner	11
3.2.1. Retrieving data from the database	11
3.2.2. Student grouping	12
3.2.3. Schedule Template Creation	14
3.2.4. Teacher Assignment Functions	15
3.2.5. Teacher Assignment Genetic Algorithm	16
3.2.6. Classroom GA functions	18
3.2.7. Size based timeslot adjustments	20
3.2.8. Wrapping the final result in an API response	20
3.3. Agentic AI Planner	22
3.3.1. Data Setup	22
3.3.2. Graph Architecture	23
3.3.3. Function calling	24
3.3.4. Token optimization	25
3.3.5. Fact-Checking the LLMs	26
3.3.6. Future Improvements	27
<b>4. CONCLUSION</b>	<b>28</b>
<b>REFERENCE LIST</b>	<b>29</b>

# 1. Introduction

B-Robots was founded in 2017 as a new player in the field of Robotic Process Automation (RPA). They offer innovative and comprehensive solutions to businesses who are burdened with repetitive, time-consuming tasks. RPA involves the use of software robots to handle rule-based digital operations, helping companies improve efficiency, reduce costs, and minimize human error.

With a mission to transform how businesses operate, B-Robots focuses on delivering smart automation systems that empower human employees to focus on more strategic and creative work. Over time, the company has significantly expanded its technological expertise. In 2019, it began integrating low-code platforms, enabling faster and more flexible application development with minimal coding. By 2020, it had introduced Intelligent Document Processing (IDP), combining OCR, machine learning, and AI to extract and process data from unstructured documents. Most recently, in 2024, B-Robots entered the next frontier of automation with Agentic Process Automation (APA).

APA introduces autonomous software agents. Digital entities that can perceive their environment, make decisions, and act independently to accomplish complex goals. These advancements reflect B-Robots' commitment to staying at the forefront of automation technology.



During my internship at B-Robots, I had the opportunity to work on a range of innovative projects that reflected the company's forward-thinking approach. One of my main contributions involved developing a planning application that leveraged genetic algorithms. A type of optimization technique inspired by the process of natural selection, used to find high-quality solutions in complex problem spaces. This project also involved implementing programmable business rules to guide automated decision-making.

Additionally, I participated in an OCR-based document automation project, helping design workflows for extracting structured data from scanned documents using Optical Character Recognition (OCR). A technology that converts images of text into machine-readable formats.

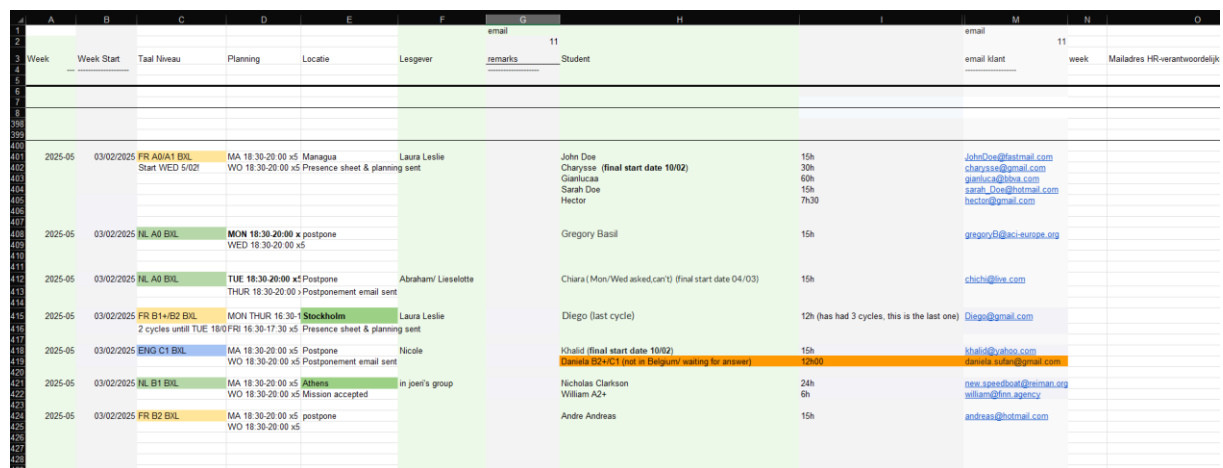
These experiences gave me practical exposure to both advanced algorithmic problem-solving and real-world automation tools, deepening my understanding of how emerging technologies are applied to optimize business operations.

## 2. Analysis

This chapter outlines the scope of the project and describes the research conducted to identify potential solutions for the core problem it aims to address. It begins by defining the specific planning challenge that the project focuses on, followed by a detailed examination of the various technologies and methodologies considered. Each proposed solution is explained and evaluated in terms of its advantages, limitations, and relevance to the project's objectives. Based on this comparative analysis, a well-supported decision is made to determine the most appropriate approach for implementing the automated planning system.

### 2.1. Project Scope

This project involves the development of a planning application for LanguageTeams, a language school that offers group lessons to students. Currently, lesson scheduling is handled manually through coordination between staff, teachers, and students. This method is time-consuming and prone to inefficiencies, especially as the number of classes and participants grows.



Week	Week Start	Taal Niveau	Planning	Locatie	Lesgever	remarks	Student	email	email klant	week	Mailadres HR-verantwoordelijk
2025-05	03/02/2025	FR A0/A1 BXL	MA 18:30-20:00 x5	Managua	Laura Leslie		John Doe Charysse (final start date 10/02) Gianluca Sarah Doe Hector	15h 30h 60h 15h 7h30	JohnDoe@testmail.com charysse@gmail.com gianluca@hiva.com sarah_doe@hotmail.com hector@gmail.com	11	
2025-05	03/02/2025	NL A0 BXL	MON 18:30-20:00 x postpone				Gregory Basil	15h	gregoryb@aci-sunrise.org		
2025-05	03/02/2025	NL A0 BXL	TUE 18:30-20:00 x Postpone		Abraham/ Lisselotte		Chiara ( Mon/Wed asked, can't ) (final start date 04/03)	15h	chich@live.com		
2025-05	03/02/2025	FR B1+B2 BXL	MON THUR 16:30-17:30 x5	Stockholm	Laura Leslie		Diego (last cycle)	12h (has had 3 cycles, this is the last one)	Diego@gmail.com		
2025-05	03/02/2025	EN/C1 BXL	MA 18:30-20:00 x5 Postpone		Nicole		Khalid (final start date 10/02) Daniela B2+K1 (not in Belgium waiting for answer)	15h 12h00	khalid@yaho.com daniela.sufan@gmail.com		
2025-05	03/02/2025	NL B1 BXL	MA 18:30-20:00 x5	Adams	in joen's group		Nicholas Clarkson William A2+	24h 6h	nrc.speelbaat@vriaman.org william@fms.agency		
2025-05	03/02/2025	FR B2 BXL	MA 18:30-20:00 x5 postpone				Andre Andreas	15h	andreas@hotmail.com		

Figure 1: Previous Solution

As can be seen in Figure 1 which is one of many excel sheets, they used to save their data in an excel spreadsheet. This brings with a lot of issues such as empty fields, data in the wrong column such as the language level in the student column instead of the language level column, usage of multiple fonts and many other issues.

LanguageTeams has requested the creation of an application that can automatically generate lesson schedules while considering various constraints and preferences. The application consists of two major components. The first component is a web interface where teachers can view their schedules and provide their general weekly availability, along with exceptions for specific dates. The staff can also use this interface to manage core elements of the system, such as adding or editing courses, assigning teachers, and managing classrooms. This part of the application is being developed by one of my teammates.

The second component, and the primary focus of this document, is the automated planning engine. This part of the system is responsible for generating valid schedules that comply with a wide range of business rules, such as group size limits, teacher availability, room assignments, and course requirements. For this component, I was given full creative freedom to choose the technologies and methodologies most suitable for solving the scheduling problem.

## 2.2. Application

This section describes the technologies and tools used in the development of the web application component of the planning system. Some tools and platforms were required as part of the project scope, while others were selected based on their suitability for the project's goals.

### 2.2.1. Django Web Framework

The core of the application was built using **Django**, a high-level Python web framework. Django was a required technology for this project, as it is the preferred framework used by the client. It supports rapid development and clean architecture through built-in features such as:

- An admin interface for managing data
- Authentication and user permissions
- A powerful ORM for database interactions
- Clear separation between backend logic and presentation templates

These features allowed for quick development of both the scheduling logic and the user-facing functionalities.

### 2.2.2. Azure Deployment

Microsoft **Azure** was another required technology, selected by LanguageTeams to host the final version of the application. Azure was chosen due to its alignment with the client's existing infrastructure and its support for scalable web services.

Azure was used to host both the web application and the backend database, specifically:

- **Azure App Services** for deploying and running the Django application
- **Azure SQL Database** for storing planning data and user information

Azure also facilitated version control and CI/CD integration, contributing to a smooth deployment pipeline.

### 2.2.3. Docker in Development

Although Azure was used for production deployment, **Docker** was leveraged during development and testing. Docker containers allowed the development environment to be consistent across devices and team members. It also made it easier to simulate production-like setups during local testing.

Docker was particularly useful for:

- Running the Django app in isolated containers
- Testing different configurations and dependencies
- Reducing environment-specific bugs

## 2.2.4. Frontend Styling: CSS and Bootstrap

For styling the application interface, a combination of CSS and Bootstrap was used. These technologies were not predetermined, and their selection followed a comparison of various frontend styling frameworks. To ensure the best fit for the project, several options were evaluated based on key criteria such as ease of use, documentation, flexibility, responsiveness, and development speed. The results of this comparison are shown in the table below:

Decision Matrix – Frontend Styling Options.

Criteria	CSS + Bootstrap	Tailwind CSS	CSS Only
Ease of Use	High	Medium	Low
Documentation & Support	High	High	Medium
Flexibility	Medium	High	High
Responsiveness	High	Medium	Low
Development Speed	High	Medium	Low

After comparing different options, Bootstrap was selected due to its:

- Ready-to-use components and grid system
- Extensive documentation and community support
- Built-in responsiveness for mobile and tablet devices

This allowed for faster interface development while maintaining a professional and consistent design. Custom CSS was also used where necessary to handle specific layout and styling requirements not covered by Bootstrap.

## 2.3. Genetic Algorithm Planner

A **Genetic Algorithm (GA)** is a nature-inspired optimization technique based on the principles of natural selection and evolution. It works by generating a population of possible solutions (called individuals), evaluating how well each one performs, and iteratively improving them through genetic operations like selection, crossover, and mutation.

In the context of this project, GAs were used to automatically create valid and optimized lesson schedules for a language school. The scheduling problem presented several complex constraints:

- Students must be grouped by course and language level.
- Each group requires a qualified teacher.
- Classrooms must be assigned for each scheduled lesson.
- Teachers and rooms must not be double-booked within the same timeslot.
- Teacher availability and date-specific exceptions must be respected.

This multi-faceted problem involves a large number of variables and potential conflicts, making brute-force or rule-based approaches impractical. A GA is well-suited for this type of problem because it can efficiently explore a vast search space and iteratively evolve toward better solutions without needing a perfect initial configuration.

Each **individual** in the population represents a complete weekly schedule, where the **genes** encode lesson assignments including teacher, group, room, and timeslot. A custom **fitness function** evaluates each schedule based on how many constraints it satisfies, penalizing invalid groupings, unavailable resources, or scheduling conflicts.

Through multiple generations, the algorithm selects the best-performing individuals, combines their traits, and introduces small random changes (mutations) to explore new possibilities. This evolutionary cycle continues until a sufficiently valid or optimized schedule is found.

To implement this approach effectively, the **DEAP** library was used due to its flexibility and support for evolutionary algorithms in Python. It provided the necessary tools to structure the GA and experiment with different genetic configurations during development.

## 2.4. Agentic AI Planner

In addition to the Genetic Algorithm-based solution, a second planner was developed using an Agentic AI approach. Agentic AI systems rely on autonomous entities, called agents, which can perceive their environment, make decisions, and interact with each other to reach a common goal. These agents simulate individual actors or components in a system and collaborate to solve complex problems. Rather than relying on a single algorithm to generate a solution, this model distributes responsibility across multiple agents that reason and act independently.

This approach is particularly suited for scheduling problems, where constraints often shift or vary depending on the context. In the case of LanguageTeams, the scheduling problem involved several layers of complexity. Students had to be grouped by course and level, teachers needed to be assigned to these groups based on availability and qualifications, classrooms had to be reserved without conflicts, and all scheduling decisions needed to respect availability constraints and avoid double bookings.

The Agentic AI planner addressed this problem by modelling each major component in the planning process as a distinct agent. Teacher agents managed their personal calendars and could respond to proposed lesson times. Group agents represented student groups seeking available teaching slots. Room agents maintained classroom schedules and accepted or declined booking requests. A central coordinator agent orchestrated the overall process, coordinating the others and evaluating whether a proposed lesson was valid.

The system was built using LangChain in combination with LangGraph. LangChain provided the infrastructure for building the logic behind each agent's decisions and memory. LangGraph was used to structure how agents interacted with one another in a flow, allowing the planner to simulate negotiations and retries when conflicts occurred. For example, when a group agent requested a lesson, the coordinator consulted the corresponding teacher and room agents. If any of them were unavailable, the coordinator tried a different timeslot or room, continuing this process until a valid solution was found or all options were exhausted.

By simulating how real-world planning often works through conversation, negotiation, and compromise. The agentic approach introduced a higher level of realism and adaptability. It was especially effective in modelling unpredictable or changing availability, where hard-coded logic would have struggled to keep up.

Although this method required a more complex setup and deeper reasoning logic, it demonstrated how multi-agent systems can be used to solve traditional planning problems in a more flexible and human-like way. It also served as an experimental contrast to the Genetic Algorithm planner, providing insight into which type of solution performed better under various scheduling conditions.



## 2.5. Conclusion

This project explored two fundamentally different approaches to solving the scheduling problem: a Genetic Algorithm (GA) and an Agentic AI-based planner. Both were developed and implemented in parallel to evaluate how each model handles the complexity of the planning task and to gain insight into their practical differences.

The Genetic Algorithm approach provided a structured and centralized optimization process. It excelled at generating complete planning solutions efficiently by exploring a large space of possible schedules. Its strengths lay in performance, repeatability, and the ability to fine-tune solutions through a custom fitness function. Once configured, it was capable of producing valid plans that respected availability constraints, avoided double bookings, and fulfilled the basic business rules.

In contrast, the Agentic AI planner took a decentralized and dynamic route. It modelled the planning process as a negotiation between independent agents, each representing a stakeholder such as a teacher, a student group, or a classroom. This method introduced flexibility and realism into the system, allowing it to adapt to changing availability or constraints without the need to reprogram a global optimization function. It also allowed for modular logic, which made it easier to extend or adjust specific parts of the system.

While the Genetic Algorithm was more efficient in generating optimized results, the Agentic AI model offered better transparency and adaptability in situations where constraints evolve over time. Each method brought valuable insights into how complex scheduling problems can be approached, and the dual development allowed for a practical comparison not just in theory, but in actual performance and user experience.

Ultimately, the goal of developing both was not to choose one over the other, but to assess their strengths in context. The experience highlighted how the nature of the problem, whether it is static and defined. Or dynamic and interactive, can determine which solution model is more appropriate.

## 3. Realisation

This chapter describes the development of the scheduling logic for the LanguageTeams planning system. The core focus was on designing and implementing two planning approaches: one based on Genetic Algorithms and the other using Agentic AI. These components operate behind the scenes of the web application and are responsible for automatically generating lesson schedules based on a variety of business rules and constraints.

The chapter begins with the preparation work required for implementing and testing the planners, followed by an in-depth look at the design and implementation of the Genetic Algorithm-based planner.

### 3.1. Data Preparation

At the start of the project, one of the client's key requests was to move away from their existing method of using Excel files to manage scheduling data. While Excel offered flexibility, it also introduced a high risk of errors, inconsistent formatting and difficulty in enforcing data integrity. These problems became even more relevant as the scheduling process grew in complexity and scale.

To address this, the decision was made to transition to a structured PostgreSQL database. I played a role in designing the structure of this database, looking at the Excel files and extracting the relevant data, this includes defining tables for courses, teachers, rooms, students, schedules as well as relationships between them that introduced additional tables such as "student\_courses", which is a table that stores all the enrolments made by students. I also thought of metadata which would be required to create a working planning algorithm.

It's important to note that while I helped create the design for the database, I was not responsible for implementing the database infrastructure itself. The actual setup and integration into the web application were handled by other members of the team.

The redesigned data structure laid the groundwork for the development and testing of the planning algorithms, enabling them to operate on clean, validated and consistent data.

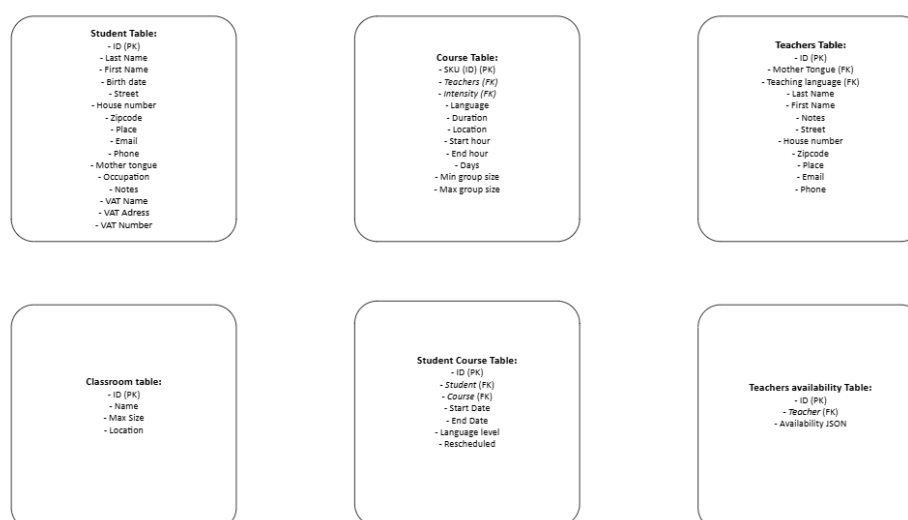


Figure 2: Database Infrastructure Drawing

**Note, this figure is not a traditional representation of a database, this is a snippet from the project architecture which includes data sources, application screen, etc...**

## 3.2. Genetic Algorithm Planner

The first scheduling engine was implemented using two consecutive Genetic Algorithms (GAs). This dual-stage approach was designed to reduce the size of the search space at each step, resulting in faster optimization and improved performance. The first algorithm focuses on assigning teachers to student groups based on availability and qualifications. Once this is complete, the output is passed to a second GA that assigns suitable classrooms to each scheduled lesson.

This layered strategy was chosen for its effectiveness in solving complex, constraint-heavy planning problems, where finding valid combinations across multiple resource types (teachers, rooms, timeslots) would otherwise be computationally expensive.

Each schedule is represented as a structured combination of student groups and their assigned resources. A single individual in the GA population reflects a complete weekly schedule, including group-teacher pairings, their timeslots, and later their classroom assignments.

The algorithm operated in the following steps:

### 3.2.1. Retrieving data from the database

Before I can start the scheduling process I have to retrieve the necessary data from the application's database through its API. This data serves as the foundation for the scheduling system and includes details such as course information, student information, classrooms, teachers, course types and if available previous scheduling data.

Once retrieved this data is processed and stored in appropriate data structures (objects) that can be easily utilized throughout the scheduling and optimization process.

The following data is needed and gets retrieved from the API:

- Course details
- Students
- Student courses
- Teachers
- Teacher Availability
- Classrooms
- Intensities (course types)
- Planning (previous schedules)

```
class Course:
    def __init__(
        self, id: int, sku: str, language_id: int, intensity_id: int, duration: int, location_id: int,
        start_hours: List[str], end_hours: List[str], days: List, min_group_size: int, max_group_size: int,
        teachers: List[Teacher]
    ):
        self.id = id
        self.sku = sku
        self.language_id = language_id
        self.intensity_id = intensity_id
        self.duration = duration
        self.location_id = location_id
        self.start_hours = start_hours
        self.end_hours = end_hours
        self.days = days
        self.min_group_size = min_group_size
        self.max_group_size = max_group_size
        self.teachers = teachers
```

Figure 3: Class example of retrieved data

These classes (objects) are then used to store the data in lists so they can be passed throughout the system in a consistent and structured manner. This approach enables efficient filtering, grouping and access during the scheduling logic and allows downstream components such as the genetic algorithm and constraint validation methods to rely on uniform, well-structured inputs.

For example a list of teacher objects can be iterated over to check availability, a list of course objects can be grouped by intensity type and previous planning entries can be referenced to ensure consistency across iterations.

By organizing the retrieved data into object-oriented representations early in the process, the entire scheduling system remains modular, extensible and easier to debug and maintain.

### 3.2.2. Student grouping

The first step in the scheduling process is grouping the students who have registered on the platform. Each group can contain a maximum of eight students. Students must be enrolled in the same course to be grouped together, and their language proficiency levels must be compatible.

To support flexibility while respecting group size constraints, it was also possible to merge smaller groups if their levels were adjacent. The accepted levels follow the Common European Framework of Reference (CEFR) progression: A0, A0+, A1, A1+, A2, A2+, B1, B1+, B2, B2+, and C1. Groups could be merged across adjacent levels (e.g., A1 and A1+), but only once. This ensured that merged groups would not span incompatible levels like A1 and B1.

To implement this logic, three deterministic functions were created. These are used sequentially to build final groupings that meet the business rules.

#### 1. Group students by course and language level

This function iterates over all active course enrolments, grouping students first by course and then by their declared language level. The result is a nested structure where each course ID maps to a dictionary of level-specific student lists. This forms the base structure for further processing.

#### 2. Split and merge groups

After initial grouping, some groups may exceed the maximum size of eight students. These oversized groups are split. The function then attempts to merge smaller groups with adjacent levels, using a defined language level order to guide which combinations are valid. Merging is only done when the resulting group size does not exceed the maximum.

#### 3. Group students by course, level, and merging

This function brings the previous two steps together to form the final student groups. Each group includes a list of one or two adjacent language levels and a list of students. Every group is also assigned a unique group ID for easy reference in later stages of the planning algorithm.

The output of this process is a collection of well-structured groups that reflect both the pedagogical structure of LanguageTeams and the technical constraints of the scheduling engine.

```
Course 7:
  Levels: A0 (1 students)
11
  Levels: A0+ (1 students)
40
  Levels: A1 + A1+ (5 students)
36
38
1
7
39
  Levels: A2 (1 students)
10

Course 8:
  Levels: A0 + A0+ (3 students)
19
2
26
  Levels: A1+ + A2 (3 students)
47
18
20
```

*Figure 4: Output of grouping algorithm*

### 3.2.3. Schedule Template Creation

Before assigning teachers or classrooms, a schedule template must first be generated. This template outlines all lessons that need to be planned, based on the course configurations and the merged student groups from the previous step.

Each course has defined start and end dates, available weekdays for lessons, and one or more possible timeslots. Some courses allow multiple weekly sessions, each with its own set of allowed days and hours. Additionally, courses differ in intensity, which influences whether certain timeslots are suitable. All of these factors must be considered when building the schedule template.

To automate this process, a new function was implemented. It loops through every active course and its associated student groups, and for each valid date between the course's start and end, it checks whether the day matches the allowed lesson days. If so, it schedules one or more lessons based on the defined timeslots. It ensures that each proposed timeslot fits the intensity requirement of the course using a helper function.

This helper function checks whether the proposed start time aligns with the course's intensity constraints. For example, private and hyper-intensive lessons are only allowed to run in the morning since the afternoon lessons for these intensities are private lessons and are not being scheduled by the algorithm as they have been deemed out of scope by B-Robots. The function parses the start time and excludes any slots beginning at or after noon.

The output of this function is a list of lessons. Each lesson is represented as a tuple containing:

- The active course
- The group ID
- The date of the lesson
- The start time
- The end time

This list acts as the foundation for further processing. It captures the complete set of sessions that need to be filled with teacher and classroom assignments. The generated schedule is also reusable and can be referenced by multiple scheduling engines or stored as a base for validation.

With the lesson structure in place, the next step is assigning teachers who are both qualified and available for each session.

### 3.2.4. Teacher Assignment Functions

Just like the student grouping, assigning teachers to groups also involves several important constraints. Teachers must be available at the scheduled time, they must be qualified to teach the assigned course, and ideally, the same teacher should be assigned to all lessons for a given group. This ensures consistency in the learning experience and simplifies both communication and lesson tracking. If no teacher is available for the full duration of a course, the algorithm tries to assign a teacher who can cover as many lessons as possible.

To handle this part of the scheduling, three functions were developed to structure the teacher assignment process. These work together to identify eligible teachers for each lesson and prioritize assignments accordingly.

#### 1. Allowed Teachers for Lesson

This function takes in a single lesson (lessons are an entry in the schedule template) and returns a list of teachers who are eligible to teach it. It first identifies which teachers are associated with the relevant course, then filters them based on their availability for the lesson's timeslot. Teachers who meet both requirements are included in the result. This narrows down the options early and ensures that the GA only considers valid assignments.

#### 2. Is Teacher Available for Timeslot

To support the filtering above, this function checks a teacher's availability for a specific lesson time. It looks at their recurring weekly schedule and any date-specific exceptions they may have entered. If a teacher has no availability data, the system assumes they are fully available. This logic allows for flexibility while also respecting the teacher's constraints.

#### 3. Prioritized Teacher Assignment

After filtering for eligibility, this function assigns teachers to lessons using a priority-based approach. If a previous version of the schedule exists (for example, from an earlier planning attempt), the algorithm first attempts to preserve those teacher assignments where valid. Otherwise, it selects from the available teachers using basic heuristics, such as assigning a teacher to as many lessons for the same group as possible to maintain consistency. If no teacher can be assigned, the function returns an error code to flag the issue for manual review or reprocessing.

This combination of constraint filtering and priority-based selection allows the Genetic Algorithm to generate schedules that are both valid and aligned with real-world teaching requirements. The output of this step is a fully teacher-assigned schedule, ready to be passed into the second Genetic Algorithm for classroom allocation.

### 3.2.5. Teacher Assignment Genetic Algorithm

Assigning teachers to a lesson schedule involves more than simply filling slots. It requires managing multiple constraints, such as availability, double-bookings, and group consistency. To navigate this complex space, we implemented a Genetic Algorithm (GA) to iteratively evolve and optimize teacher assignments.

The core idea behind a GA is to mimic natural selection: a population of candidate solutions (schedules) is generated and refined over several generations. Each candidate, or individual, represents a full teacher assignment for all lessons in the schedule. The algorithm evaluates how good each individual is using a fitness function, and through processes like selection, crossover, and mutation, the population evolves toward better solutions (i.e., schedules with fewer conflicts).

The GA setup begins with the setup function, which configures the DEAP toolbox. It initializes the data structures, defines the objective as conflict minimization, and generates individuals based on the prioritized teacher assignment function. It also caches allowed teachers per lesson to improve performance during evaluation.

Each generation follows a typical GA cycle. A new population is first created through selection, where better-performing individuals are chosen to pass on their characteristics. In our case, we use tournament selection, which randomly picks a small group of individuals and selects the best among them, this helps balance exploration with preserving high-quality solutions. Next, crossover (also called recombination) is applied between pairs of individuals, combining segments of their teacher assignments to produce new offspring. This mimics genetic inheritance and introduces variation. Additionally, mutation is used to randomly change individual gene values by reassigning a lesson to a different teacher, to maintain diversity in the population and avoid local optima.

Given that these operations can produce infeasible schedules (e.g., assigning the same teacher to overlapping lessons), a repair function is used to resolve conflicts. This function attempts to reassign problematic lessons to other valid teachers. If no suitable reassignment is found, a dummy teacher placeholder is used to flag unresolved conflicts. The repair step is essential to prevent the GA from degrading over time due to the accumulation of invalid assignments.

The fitness function (evaluate) assigns a score to each individual based on the total number of conflicts. It penalizes three main issues:

- Assigning teachers who are not eligible for a given lesson.
- Double-booking the same teacher in the same timeslot.
- Inconsistent teacher assignments within the same student group.

By minimizing these penalties, the GA favours assignments that are both valid and pedagogically coherent.

Across generations, the GA tracks statistics such as average and minimum fitness to monitor progress. The best-performing individual is preserved in a hall of fame and returned at the end of the run, along with a full log of the evolution.



```

Assigned Lessons Per Teacher:

Teacher: Olivia Lopez
  Course 1, Group 1, Date 2025-03-10, Timeslot 09:30:00-11:30:00
  Course 1, Group 1, Date 2025-03-10, Timeslot 12:30:00-14:30:00
  Course 1, Group 1, Date 2025-03-11, Timeslot 09:30:00-11:30:00
  Course 1, Group 1, Date 2025-03-11, Timeslot 12:30:00-14:30:00
  Course 1, Group 1, Date 2025-03-12, Timeslot 09:30:00-11:30:00
  Course 1, Group 1, Date 2025-03-12, Timeslot 12:30:00-14:30:00
  Course 1, Group 1, Date 2025-03-13, Timeslot 09:30:00-11:30:00
  Course 1, Group 1, Date 2025-03-13, Timeslot 12:30:00-14:30:00
  Course 1, Group 1, Date 2025-03-14, Timeslot 09:30:00-11:30:00
  Course 1, Group 1, Date 2025-03-14, Timeslot 12:30:00-14:30:00

Teacher: Lucas Gonzalez
  Course 1, Group 2, Date 2025-03-10, Timeslot 09:30:00-11:30:00
  Course 1, Group 2, Date 2025-03-10, Timeslot 12:30:00-14:30:00
  Course 1, Group 2, Date 2025-03-11, Timeslot 09:30:00-11:30:00
  Course 1, Group 2, Date 2025-03-11, Timeslot 12:30:00-14:30:00
  Course 1, Group 2, Date 2025-03-12, Timeslot 09:30:00-11:30:00
  Course 1, Group 2, Date 2025-03-12, Timeslot 12:30:00-14:30:00
  Course 1, Group 2, Date 2025-03-13, Timeslot 09:30:00-11:30:00
  Course 1, Group 2, Date 2025-03-13, Timeslot 12:30:00-14:30:00
  Course 1, Group 2, Date 2025-03-14, Timeslot 09:30:00-11:30:00
  Course 1, Group 2, Date 2025-03-14, Timeslot 12:30:00-14:30:00

Teacher: Michael Johnson
  Course 2, Group 1, Date 2025-03-11, Timeslot 09:30:00-11:30:00
  Course 2, Group 1, Date 2025-03-13, Timeslot 09:30:00-11:30:00
  Course 2, Group 1, Date 2025-03-18, Timeslot 09:30:00-11:30:00
  Course 2, Group 1, Date 2025-03-20, Timeslot 09:30:00-11:30:00

```

*Figure 5:Teacher Scheduling Output*

This approach provides a flexible and robust solution to the teacher assignment problem. It balances structural constraints with randomness and repair mechanisms, allowing it to explore a wide space of possibilities while maintaining realistic and usable schedules.

### 3.2.6. Classroom GA functions

After finalizing teacher assignments, the next stage in the scheduling process involved allocating classrooms to each lesson. While this task may appear logistical, it presents several non-trivial challenges: Ensuring classrooms meet group size requirements, avoiding scheduling conflicts such as double-bookings and maintaining room consistency for student groups across their lessons. To address this, I extended the Genetic Algorithm to also handle classroom assignments in a similarly robust and automated way.

As with teacher assignments, each classroom solution is represented as an individual in the GA: a list of indices where each index corresponds to a specific classroom assigned to a given lesson. The algorithm evaluates these individuals based on how well they satisfy room-related constraints, using a dedicated fitness function.

The fitness function for classroom assignments penalizes schedules based on several factors:

- **Capacity violations:** If a group is assigned to a classroom that is too small, a penalty is applied.
- **Double bookings:** Lessons scheduled in the same classroom at the same time incur a significant penalty.
- **Group Consistency:** Assignments are penalized if the same group uses different classrooms throughout their course, encouraging stability and convenience.
- **Classroom Utilization:** Small mismatches between classroom capacity and group size are penalized lightly, while perfect fits are rewarded.

This encourages the algorithm to favour practical, conflict-free room allocations that minimize unnecessary movement between classrooms, increasing the convenience for all students.

#### Initialization

To seed the GA with sensible starting points, an initialization function assigns each lesson to the smallest classroom that can accommodate the group, while also considering previous classroom assignments if available. This helps maintain continuity across schedules and improves convergence by starting from feasible layouts.

#### Mutation

To maintain diversity in the population and avoid premature convergence a mutation function was added, this function introduces random changes to classroom assignments. Each lesson has a small probability (indpb) of being reassigned to a different valid room. After mutation a repair function is triggered to ensure no invalid states are introduced.

#### Repair

The repair function ensures that classroom assignments are valid post-crossover or mutation. It resolves double bookings, prioritizes consistency for each group, and automatically assigns online classrooms to a designated "ONLINE" classroom. If no valid rooms are found a fallback dummy classroom is used.

The repair mechanism plays a crucial role in maintaining the integrity of the population and enables the GA to operate effectively even in tightly constrained scenarios.

Assigned Lessons Per Teacher:					
Teacher: Olivia Lopez					
Course 1, Group 1, Date	2025-03-10, Timeslot	09:30:00-11:30:00	MANAGUA (6)		
Course 1, Group 1, Date	2025-03-10, Timeslot	12:30:00-14:30:00	MANAGUA (6)		
Course 1, Group 1, Date	2025-03-11, Timeslot	09:30:00-11:30:00	MANAGUA (6)		
Course 1, Group 1, Date	2025-03-11, Timeslot	12:30:00-14:30:00	MANAGUA (6)		
Course 1, Group 1, Date	2025-03-12, Timeslot	09:30:00-11:30:00	MANAGUA (6)		
Course 1, Group 1, Date	2025-03-12, Timeslot	12:30:00-14:30:00	MANAGUA (6)		
Course 1, Group 1, Date	2025-03-13, Timeslot	09:30:00-11:30:00	MANAGUA (6)		
Course 1, Group 1, Date	2025-03-13, Timeslot	12:30:00-14:30:00	MANAGUA (6)		
Course 1, Group 1, Date	2025-03-14, Timeslot	09:30:00-11:30:00	MANAGUA (6)		
Course 1, Group 1, Date	2025-03-14, Timeslot	12:30:00-14:30:00	MANAGUA (6)		
Teacher: Lucas Gonzalez					
Course 1, Group 2, Date	2025-03-10, Timeslot	09:30:00-11:30:00	TOKYO (6)		
Course 1, Group 2, Date	2025-03-10, Timeslot	12:30:00-14:30:00	TOKYO (6)		
Course 1, Group 2, Date	2025-03-11, Timeslot	09:30:00-11:30:00	TOKYO (6)		
Course 1, Group 2, Date	2025-03-11, Timeslot	12:30:00-14:30:00	TOKYO (6)		
Course 1, Group 2, Date	2025-03-12, Timeslot	09:30:00-11:30:00	TOKYO (6)		
Course 1, Group 2, Date	2025-03-12, Timeslot	12:30:00-14:30:00	TOKYO (6)		
Course 1, Group 2, Date	2025-03-13, Timeslot	09:30:00-11:30:00	TOKYO (6)		
Course 1, Group 2, Date	2025-03-13, Timeslot	12:30:00-14:30:00	TOKYO (6)		
Course 1, Group 2, Date	2025-03-14, Timeslot	09:30:00-11:30:00	TOKYO (6)		
Course 1, Group 2, Date	2025-03-14, Timeslot	12:30:00-14:30:00	TOKYO (6)		
Teacher: Michael Johnson					
Course 2, Group 1, Date	2025-03-11, Timeslot	09:30:00-11:30:00	BERLIN (6)		
Course 2, Group 1, Date	2025-03-13, Timeslot	09:30:00-11:30:00	BERLIN (6)		
Course 2, Group 1, Date	2025-03-18, Timeslot	09:30:00-11:30:00	BERLIN (6)		
Course 2, Group 1, Date	2025-03-20, Timeslot	09:30:00-11:30:00	BERLIN (6)		

Figure 6: Classroom Assignment Output

### 3.2.7. Size based timeslot adjustments

While scheduling lessons, not all time allocations need to be rigid (two hours long by default). In certain cases, groups smaller than three, less time is allocated. To make accommodations for these changes I introduced a post-processing step that adjusts lesson times based on the size of the assigned group and the time of day. These adjustments were introduced to comply with a predefined rule provided by the client, without additional context.

The adjustments depend on both group size and time of day:

- Morning (before 12:00):
  - Group size one: start time delayed by one hour
  - Group size two: start time delayed by thirty minutes
- Afternoon or evening:
  - Group size one: End time moved by one hour.
  - Group size two: End time moved by thirty minutes

These adjustments are deterministic and applied directly to the schedule after initial generation and optimization, ensuring that group-specific timing modifications are respected.

For the implementation of this rule a “post\_process\_timeslot” function was added. This function receives a lesson tuple along with the associated group size, then identifies whether the lesson occurs in the morning and afternoon and adjusts the relevant start or end time accordingly.

### 3.2.8. Wrapping the final result in an API response

Once the optimization process for the classroom scheduling is completed using the genetic algorithms, the final results need to be formatted and stored in a way that can be easily used by the application. The scheduling information, which includes teacher assignments, classroom allocations and student group details, is wrapped in an API response. This allows the application to retrieve the data, save it in the database and display the schedule to the end users.

For this I created a function “save\_planning” which will combine the outputs of both the GA’s and format it into the following format:

```
def save_planning(planning_objects):
    response = [
        {
            "teacher_id": p["teacher_ids"],
            "classroom_id": p["classroom_id"],
            "date": p["date"],
            "start_time": p["start_time"],
            "end_time": p["end_time"],
            "student_courses": p["students"] if "students" in p and p["students"] else []
        }
        for p in planning_objects
    ]
    return {"planning": response}
```

Figure 7: API response format

This format is expected to be returned by the API call so that the application can store these planning objects in the database to facilitate further use, such as displaying the schedule on the application interface. The database interactions will typically happen in the backend of the application.

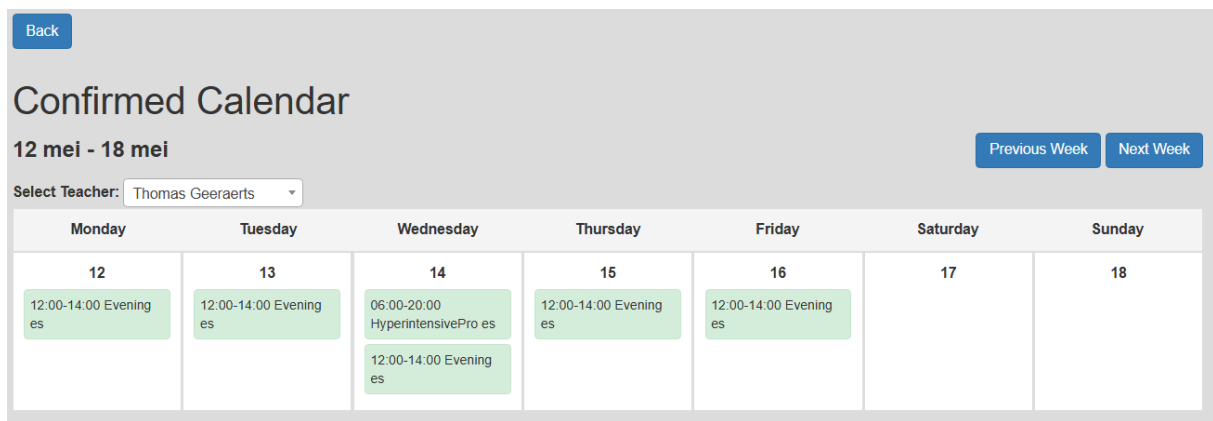


Figure 8: Schedule display in the application

## 3.3. Agentic AI Planner

The first step in exploring a more flexible and dynamic approach to scheduling was experimenting with a Large Language Model (LLM)-based planner. The idea was to see how far a language model could go in generating feasible schedules using reasoning abilities instead of hard-coded optimization logic. While the initial goal was to create a multi-step agentic system capable of tool use, time constraints led to a simplified version using structured prompts and function-calling. Despite this, it provided valuable insight into how language models could contribute to planning tasks and opened the door for future expansion into full agent-based workflows. This chapter will explore what was realised in time.

### 3.3.1. Data Setup

The first step in building the LLM-based scheduling system was preparing the data in a format that could be passed between different planning steps. This was done by defining a central state object, which represents the current status of the scheduling process at any given point in time.

A state is essentially a container that holds all relevant information needed for planning, including inputs (such as available teachers and classrooms) as well as intermediate outputs (like scheduled lessons or assigned groups). In this project, the state was modelled as a dictionary with clearly defined fields and data types using Python's *TypedDict*.

The use of a centralized state object is critical for several reasons:

- **Persistence:** It ensures that the planning context (e.g., what has already been scheduled, what resources remain) is retained across multiple function calls.
- **Modularity:** It allows different planning steps—such as grouping students, assigning teachers, or picking classrooms—to access and update shared data without depending directly on each other.
- **Orchestration compatibility:** Frameworks like LangGraph or LangChain agents rely on the concept of a state to manage transitions between different nodes in a workflow.

```
1 # LangGraph requires a class for the state
2 class ScheduleState(TypedDict):
3     days: List[Dict[str, Any]]
4     student_courses: List[Dict[str, Any]]
5     teachers: List[Dict[str, Any]]
6     teacher_availabilities: List[Dict[str, Any]]
7     courses: List[Dict[str, Any]]
8     intensities: List[Dict[str, Any]]
9     classrooms: List[Dict[str, Any]]
10    groups: List[Dict[str, Any]]
11    teacher_schedule: List[Dict[str, Any]]
12    classroom_assignments: List[Dict[str, Any]]
13    usages: List[Dict[str, Any]]
14
15 schedule_state: ScheduleState = {
16     "days": input_data.get("days", []),
17     "student_courses": input_data.get("student_courses", []),
18     "teachers": input_data.get("teachers", []),
19     "teacher_availabilities": input_data.get("teacher_availabilities", []),
20     "courses": input_data.get("courses", []),
21     "intensities": input_data.get("intensities", []),
22     "classrooms": input_data.get("classrooms", []),
23     "groups": [],
24     "teacher_schedule": [],
25     "classroom_assignments": [],
26     "usages": [],
27 }
```

Figure 9: State object

### 3.3.2. Graph Architecture

Following the data setup phase, where the initial state is constructed to include the course, student, and availability information, the next step involves structuring the scheduling logic using a graph-based approach. This is implemented using LangGraph, which enables chaining modular functions into a directed graph that controls how the state is passed and modified across steps.

Rather than executing a strict linear pipeline, the LangGraph setup defines a more flexible and interpretable flow. Each node in the graph corresponds to a specific function such as grouping students, generating the schedule template, or assigning teachers. These functions take in the shared state, update it with new data, and return the modified version. The edges between these nodes define the execution path and ensure the right order of operations.

The graph is initialized with the state containing all pre-loaded data. This state is then passed from node to node. For example, the "generate student groups" function reads the course and enrolment information from the state and adds a new entry with the generated groups. The next node might generate an initial schedule layout based on those groups, and so on. At each stage, the state becomes richer and more complete, until it contains all the information needed to produce the final schedule.

This graph-driven structure not only improves clarity and modularity but also makes experimentation much easier. Since each part of the process is isolated in its own node, components can be swapped or extended without disrupting the entire system. It also lays the groundwork for more complex behaviour in the future. Such as allowing a node to decide dynamically which path to follow next, or integrating tool-using agents inside individual nodes.

```
1 graph = StateGraph(state_schema=ScheduleState)
2 graph.add_node("grouping", update_state_with_groups)
3 graph.add_node("teacher_agent", update_state_with_teacher_assignments)
4 graph.add_node("classroom_agent", update_state_with_classroom_assignments)
5
6 graph.add_edge(START, "grouping")
7 graph.add_edge("grouping", "teacher_agent")
8 graph.add_edge("teacher_agent", "classroom_agent")
9 graph.add_edge("classroom_agent", END)
10
11
12 workflow = graph.compile()
13 initial_state = schedule_state.copy()
14 final_state = workflow.invoke(initial_state)
```

Figure 10: Graph Implementation

### 3.3.3. Function calling

Instead of using autonomous agents, structured function-calling was implemented as a lightweight alternative. For each core task (e.g., generating a teacher schedule or assigning classrooms), a dedicated function was defined. These functions were invoked via system-level prompts instructing the LLM to fill in missing schedule data based on the provided constraints and the current state. These prompts were also given data from the state whenever they were called.

This approach kept control of the logic while leveraging the LLM's reasoning for specific tasks, like:

- Distributing lessons across available days
- Assigning teachers based on availability
- Ensuring room capacities weren't exceeded
- Grouping students by course and level

While less flexible than full agentic behaviour, this setup allowed for more predictable execution and lower token usage.

```
# Format the prompt with the provided data
prompt = ChatPromptTemplate.from_messages([
    [
        SystemMessagePromptTemplate.from_template(
            """
            You are a scheduling assistant. Your task is to group students into groups based on their proficiency levels and
            the course they are taking. You MUST follow these rules:

            **Rules:**
            1. Each group has a maximum of 8 students.
            2. Smaller groups than 3 are allowed but not preferred.
            3. All students in a group must be enrolled in the same course.
            4. All students in a group must have the same proficiency level, unless rule 5 applies.
            5. A group can only contain students from one level, or from exactly two levels that are directly next to each other in the language levels list.
            | You CANNOT combine three or more levels in a single group, even if they are all adjacent.
            6. You can only combine students who are enrolled in the same course.
            7. Each student must be referenced only by their 'id' field (not 'student_id'). Do not use or invent any other IDs.
            8. All students must be assigned to a group. Do not leave any student unassigned.
            9. The only allowed language levels are in the Language Levels list. Do not use any other levels.
            10. Output only the groups as a list of dictionaries in the specified format. Do not include any explanation or extra text.

            Given the following data, create groups of students based on the rules and output the groups in the specified format:
            - Students: {student_courses}
            - Language Levels: {language_levels}

            Do NOT stray from the rules.

            output format:
            [
                [
                    {
                        "group_id": 1,
                        "course_id": "12",
                        "ids": [1, 10, 25, 50]
                    },
                    ...
                ]
            ]
            """
        ),
    ],
])

).format_messages(
    student_courses=state["student_courses"],
    language_levels=["A0", "A0+", "A1", "A1+", "A2", "A2+", "B1", "B1+", "B2", "B2+", "C1", "C1+", "C2"]
)
```

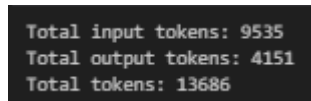
Figure 11: Function-Calling Prompt



### 3.3.4. Token optimization

When working with large language models like GPT, every word, punctuation mark, and element of structured data contributes to the total number of *tokens*. Tokens are the smallest units the model processes, for example, a word like "planning" might be one token, while "2025-06-01T09:00:00" could be many. Since models like GPT-4 have limits on the number of tokens they can process in a single request optimizing the token usage becomes crucial, especially in data-heavy applications like planning.

In this project, the data being sent to the LLM isn't that much since this is still a small scale company that will be using the solution however if they expand in the future the increased amount of data could become an issue. Besides the token limit of the model using tokens also costs money so it's important to minimize to cost where possible.



```
Total input tokens: 9535
Total output tokens: 4151
Total tokens: 13686
```

Figure 12: Token cost per iteration

To address this, I implemented token optimization strategies to ensure that only relevant and minimal information is passed to the model. This involved:

- **Filtering unnecessary data:** Only lessons that had not yet been scheduled or that required re-evaluation were included in the prompt. Any redundant metadata or previously processed entries were removed.
- **Simplifying representations:** Instead of passing full object structures, data was serialized into lean dictionaries or lists containing only essential identifiers.

Another optimization strategy I didn't get around to doing but would like to include is transforming the input and output data into a more natural language which would make the token cost much lower, using this strategy would however require additional functions to transform the data to the natural language state and back to the normal state after receiving the output of the LLM.

### 3.3.5. Fact-Checking the LLMs

One of the challenges of working with large language models for decision-making or planning is their tendency to hallucinate. Hallucinating means that they can generate plausible-sounding information but that information is not necessarily grounded in the input data. While LLMs are excellent at synthesizing patterns and reasoning over structured inputs, they can get confused or create outputs that look correct on the surface level but are factually incorrect.

In the context of this project, hallucinations or confusion can occur in various ways:

- Assigning a teacher who is not actually available
- Scheduling a lesson outside of the constraints of the timeslots
- Double booking classrooms or teachers
- Incorrectly grouping students based on their level or maximum group size

To catch these issues I designed a set of validation functions that act as a safeguard after the LLM has made its scheduling suggestions. These functions don't attempt to correct the LLM directly but instead verify whether the proposed schedule matches the rules that have been given to the LLM. This acts as a form of quality control and also lets me find out pain points in the generation of the LLM.

Fact-checking is especially important in hybrid systems like this one, where LLMs supplement algorithmic approaches. It creates a safety net that allows for more flexible, generative planning without sacrificing correctness.

In future work, these validations could be expanded into more intelligent feedback loops, where failed suggestions help fine-tune prompts or guide the model's reasoning using chain-of-thought or constraint-reflection techniques.

```
Group 3 combines non-adjacent or more than two proficiency levels: ['A0+', 'A0', 'A1']
Group 5 combines non-adjacent or more than two proficiency levels: ['A1', 'A0+', 'A0']
Group 8 combines non-adjacent or more than two proficiency levels: ['A1+', 'A2', 'A1']
Group 9 combines non-adjacent or more than two proficiency levels: ['A2', 'A1', 'A0']
Group 10 combines non-adjacent or more than two proficiency levels: ['A0+', 'A2', 'A0']
Group 11 combines non-adjacent or more than two proficiency levels: ['A0+', 'A2', 'A1+']
Group 12 combines non-adjacent or more than two proficiency levels: ['A0+', 'A1', 'A0']
Group 13 combines non-adjacent or more than two proficiency levels: ['A1+', 'A1', 'A2', 'A0+']
StudentSource IDs not assigned to any group: ['127', '121']
```

Figure 13: Example of hallucination issues

### 3.3.6. Future Improvements

While the current implementation demonstrates a working integration of large language models (LLMs) through function calling, it remains a foundational prototype. To make this system more powerful and autonomous, future iterations should move away from this basic approach and transition toward a [Model Context Protocol \(MCP\)](#) framework combined with true agent-based architectures.

The Model Context Protocol offers a way to structure the input and evolving state more explicitly, enabling the model to reason across multiple turns with awareness of prior decisions and constraints. By formalizing how data is passed, tracked, and updated during planning, MCP allows for better continuity, reduced redundancy, and clearer context management within the LLM workflow. This stands in contrast to function-calling methods, where each request typically starts with a fresh prompt and lacks memory of earlier logic or outputs.

In addition to structured context, adopting agentic behaviour is a critical step toward a more intelligent planning system. Rather than relying on a single monolithic call to produce a result, a true agent-based model would:

- Maintain and update internal state across steps.
- Use specialized tools (e.g., validators, schedulers, availability checkers) when needed.
- Evaluate its own outputs, retry failed actions, and decompose problems into subtasks.
- Decide dynamically how to proceed depending on current context and goals.

This shift empowers the system to reason over time, coordinate multiple constraints, and interact with external data sources or rule systems as needed, giving it capabilities essential for solving complex, real-world scheduling problems.

Another major area for improvement is mitigating **hallucinations**, where the LLM may generate incorrect or fabricated content. While the current system includes external validation functions to detect and reject flawed outputs, this is inherently reactive. A more robust future system would:

- Leverage MCP to make state and constraints explicit and verifiable during each step.
- Integrate fact-checking tools as callable agents that cross-check model assertions.
- Reduce token ambiguity and improve grounding through schema-constrained output formats.

By combining MCP, agentic planning, and enhanced hallucination mitigation, the system would evolve into a far more resilient and intelligent planner—capable not just of generating solutions, but of iteratively validating, adjusting, and justifying them within a dynamic, tool-assisted environment.

## 4. Conclusion

This project set out to design and evaluate two distinct approaches to solving a complex scheduling problem: a **Genetic Algorithm (GA)**-based planner and a more exploratory **Agentic AI-based planner** using large language models (LLMs). The core objective was to develop a system capable of generating valid and optimized schedules, handling real-world constraints such as teacher availability, classroom capacity, and course types, while also exploring the potential of more modern, LLM-driven planning techniques.

### RETROSPECTIVE AND KEY FINDINGS

Throughout the thesis, the Genetic Algorithm approach proved to be a robust and reliable method. It systematically generates schedules through iterative optimization, ensuring that constraints are explicitly handled and that the resulting schedules are structurally sound. This method was more complex to implement in terms of rule logic but also far more predictable and controllable.

In contrast, the Agentic AI planner, while innovative, presented more challenges. Although the function-calling mechanism enabled the LLM to dynamically respond to scheduling problems using structured prompts and validation functions, the overall correctness and consistency of its results remained a major concern. Issues such as hallucinations and token limitations required additional layers of validation logic and careful prompt engineering.

Despite these limitations, the exploration of the Agentic AI approach provided valuable insights into the potential for future systems that combine human-like reasoning with structured problem-solving. The research also highlighted how these tools can be made more powerful through techniques like Model Context Protocols (MCP), graph-based state tracking, and dynamic tool selection.

### EVALUATION AGAINST OBJECTIVES

The initial goals, to build a functioning scheduling system and explore LLM-based planning were successfully met. The GA-based system was able to generate schedules that align with real-world constraints and business rules. The agentic side, though not deployed in a production setting, served as a proof of concept, demonstrating how LLMs might eventually contribute to solving complex logistical problems.

### FUTURE OUTLOOK AND RECOMMENDATIONS

Looking forward, the agentic approach could be significantly improved by incorporating true agent frameworks that support tool selection, longer memory, and better control over hallucinations. The use of MCP and formal state representations can also help reduce token usage and bring more structure to LLM reasoning. Hybrid models, where LLMs assist with subtasks such as planning suggestions, exception handling, or natural language reporting, could also strike a balance between flexibility and reliability. However, it is important to acknowledge a fundamental limitation: scheduling is an [NP-hard](#) problem. This means there is no known algorithm that can always find an optimal solution efficiently, especially as complexity increases. As such, **there is unlikely to ever be a truly “perfect” schedule**, particularly in dynamic, real-world environments with unpredictable human factors.

### FINAL NOTE

For this reason, **human oversight and validation will always remain a crucial part of the scheduling process**. Automated systems no matter how advanced should be viewed as tools to assist human decision-makers, not to replace them. This project has demonstrated that while algorithmic and agentic methods each have their strengths, the most practical and trustworthy results today still come from structured, deterministic systems like the Genetic Algorithm planner all the while avoiding the costs associated with using LLMs. However, it is more logically complex and time consuming to create a Genetic Algorithm planner.

# Reference List

- Aston, J. (n.d.). *The efficient use of tokens for multi-agent systems*. Retrieved from capgemini: <https://www.capgemini.com/insights/expert-perspectives/ai-lab-the-efficient-use-of-tokens-for-multi-agent-systems/>
- DEAP documentation. (n.d.). Retrieved from deap: <https://deap.readthedocs.io/en/master/documentation>
- documentation. (n.d.). Retrieved from LangChain: <https://python.langchain.com/docs/introduction/introduction>
- introduction. (n.d.). Retrieved from modelcontextprotocol: <https://modelcontextprotocol.io/introduction>
- Mallawaarachchi, V. (n.d.). *introduction to genetic algorithms*. Retrieved from medium: <https://medium.com/data-science/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3>
- Show, R. (n.d.). *LangGraph Tutorial: A Comprehensive Guide for Beginners*. Retrieved from futuresmart: <https://blog.futuresmart.ai/langgraph-tutorial-for-beginners>
- Udemy. (n.d.). Retrieved from ultimate beginners guide to genetic algorithms: <https://www.udemy.com/course/the-ultimate-beginners-guide-to-genetic-algorithms-in-python/learn/lecture/28615654#overview>
- Ulindala, P. R. (n.d.). *The Hidden Costs of Agent-Driven LLMs: A Practical Analysis*. Retrieved from medium: [https://medium.com/@pawanreddy\\_u/the-hidden-costs-of-agent-driven-llms-a-practical-analysis-e1cfe1e67538](https://medium.com/@pawanreddy_u/the-hidden-costs-of-agent-driven-llms-a-practical-analysis-e1cfe1e67538)
- Weizhe Ren, Y. Q. (n.d.). *Alpha Mining and Enhancing via Warm Start Genetic Programming for Quantitative Investment*. Retrieved from arxiv: <https://arxiv.org/html/2412.00896v1>
- Why LangGraph. (n.d.). Retrieved from LangChain: <https://langchain-ai.github.io/langgraph/concepts/why-langgraph/>