



UNIVERSITATEA DIN
BUCUREȘTI

FACULTATEA DE
MATEMATICĂ ȘI
INFORMATICĂ



SPECIALIZAREA INFORMATICĂ

Disertație

**GROUP TRAVEL
RECOMMENDATION SYSTEM:
OPTIMIZING SATISFACTION FOR
EVERY INDIVIDUAL**

Absolvent

Ionescu Ariana-Georgiana

Coordonator științific

Conf. dr. Marius Popescu

București, iulie 2025

Rezumat

Planificarea unei călătorii poate să fie un proces copleșitor și îndelungat, cauzat de cantitatea mare de informații disponibile. Această disertație prezintă un sistem de recomandări de călătorie cu o interfață interactivă, ce ajută la generarea unui itinerariu personalizat pe baza preferințelor utilizatorului. Sistemul folosește generarea augmentată prin regăsire pentru a sugera activități și folosește date colectate cu ajutorul *OpenTripMap API* și *Google Places API*. Sistemul rezultat adaptează dinamic recomandările, oferind primele k atracții, clasificate în funcție de cererile utilizatorului și rafinate prin interacțiunea conversațională cu interfață interactivă. Această abordare demonstrează potențialul modelelor lingvistice pentru îmbunătățirea planificării călătoriilor prin recomandări interactive personalizate.

Abstract

Travel planning can be time-consuming and overwhelming due to the vast amount of available information. This dissertation presents a chatbot interface travel recommendation system that generates personalized itineraries based on user preferences. The system employs a Retrieval-Augmented Generation approach to suggest activities, utilizing data collected from *OpenTripMap* and *Google Places API*. The resulting system dynamically adapts the recommendations, providing top- k attractions ranked by the user's preferences and refined through conversational interaction. This approach demonstrates the potential of language models for enhancing travel planning through personalized interactive recommendations.

Contents

1	Introduction	5
2	Literature review	6
3	System Design and Methodology	8
3.1	Overview	8
3.2	Data Collection	8
3.3	Recommendation Model	9
3.3.1	Recommendation Model for a Single Individual	9
3.3.2	Recommendation Model for a Group	10
3.4	Scheduling Algorithm	11
3.5	Interactive Interface	12
4	Implementation	14
4.1	Overview	14
4.2	System Architecture	14
4.3	Technology Stack	15
4.4	Implementation Details	16
4.4.1	Data Collection	16
4.4.2	Data Processing	16
4.4.3	Recommendation System	17
4.4.4	Itinerary Generation	18
4.4.5	Interactive Interface	19
4.4.6	Output Example	20
5	Results and Evaluation	22
5.1	Information Extraction	22
5.2	System Evaluation	23
5.2.1	Point System	23
5.2.2	User Survey	24

6	Discussion	26
6.1	Strengths and Limitations of the Implementation	26
6.2	Implications and Applications	26
7	Future Work and Conclusion	28
	Bibliography	30

Chapter 1

Introduction

Planning a trip can be an overwhelming task. Trying to balance multiple constraints such as location or activity preferences, opening hours for each location, making optimal itineraries, and the variety of available attractions. With the rise of personalized services on digital platforms, users increasingly expect intelligent systems to assist in decision-making, particularly in domains such as travel, where flexibility and personalization are crucial. A travel recommendation tool that takes user preferences by processing natural language, asks leading questions to obtain all the necessary data, and is capable of making changes could be a great, accessible tool for a variety of users.

This project addresses the gap in travel recommendation systems for a group of people, but the system is designed to handle both group and individual planning. It utilizes data from public APIs (OpenTripMap, Google Places, etc.) to gather relevant information about tourist attractions across Romania. The system's implementation required combining several components, including API data retrieval, extracting user preferences, obtaining top recommendations, and scheduling itineraries. Throughout the development, emphasis was placed on modularity and extensibility of the code to allow for future improvements, such as integrating real-time traffic data, expanding the chatbot's language capabilities, or enhancing the itinerary generator's computational efficiency.

Ultimately, this work contributes to the growing trend of AI-driven recommendation systems by introducing a flexible, user-friendly interface for travel planning. It demonstrates how combining natural language processing with structured data and scheduling algorithms can result in an engaging and efficient user experience. The project also provides insights into practical challenges, such as handling incomplete user input, managing group preferences, and ensuring that the generated plans are realistic and possible.¹

¹The code is available at: https://1drv.ms/f/s!ArrAnLSZj9RTgUDSmsmEot_Svuwl?e=uiRI5d

Chapter 2

Literature review

This chapter reviews the relevant literature on travel recommendation systems, including key concepts, existing solutions, and gaps in existing research. It sets the foundation for the system design discussion in Chapter 3.

In the paper Halder et al. [9], the authors conduct a literature survey of itinerary recommendations. They compared different types of systems based on various categories. The two main divisive factors are User Satisfaction and Provider Satisfaction. User satisfaction refers to how well the recommendation system meets the needs, preferences, and expectations of the traveler, with the goal of increasing user engagement. Provider satisfaction refers to the extent to which the system supports the business goals of the service provider, enabling them to maximize sales and derive benefits from the platform. The user satisfaction is further divided into two categories: Non-personalized and Personalized. Based on these decisions, the recommendations are made. The recommendations can also be of multiple types: itinerary, top-k, or next recommendation. Different constraints can be added for the recommended type chosen.

During this survey, it was found that there is a gap in recommendations for a group of people, not just one individual, where each person receives maximum satisfaction. However, this paper was published at the end of 2023, and further research has been conducted in this domain.

In the paper Banerjee, Satish, and Wörndl [1], published in 2025(the second version), a method is presented that integrates a Retrieval-Augmented Generation system (RAG) and a Sustainability-Augmented Reranking mechanism into the tourism recommender system. The primary goal is to provide suggestions and reduce hallucinations commonly found in the outputs of large language models. The approach consists of three stages: retrieval (fetching relevant documents from the database), generation (using an LLM to generate suggestions and explanations), and sustainability-augmented reranking

(reordering the LLM's outputs based on the sustainability dimension). The authors show the improved quality of their recommendations compared to traditional systems and RAG without reranking. For evaluation, both automated metrics(relevance) and human evaluation were used.

A paper from 2024, Qi et al. [13], focuses on a RAG-based system tailored for tourism in Tibet. The goal of the research was to reduce hallucinations and improve personalization for each recommendation in culturally rich but data-scarce regions. Their approach consists of three steps: constructing the dataset and indexing it using vector embeddings, utilizing the index to retrieve the top-k relevant locations for each user, and passing the retrieved data through a large language model (LLM) to generate recommendations. The model was fine-tuned with prompts that focused on cultural sensitivity and travel constraints specific to Tibet. The authors also utilize a query refinement algorithm that reformulates vague queries to enhance the accuracy of the retrieval step. The outputs are evaluated using human evaluation.

Taking into consideration all the information previously presented, this dissertation paper focuses on personalized user satisfaction recommendations for both groups and individuals using a RAG system tailored for tourism in Romania.

Chapter 3

System Design and Methodology

3.1 Overview

This chapter serves as a guide to the system design, methods used and attempted, as well as the datasets used to achieve the final results. The main steps of this system involve data collection, retrieving the most suitable suggestions for the user, and creating an itinerary tailored to the user's preferences. The process of obtaining the final result is made more user-friendly with the help of an interface, through which the user can interact to get the final trip itinerary.

For the points of interest (POIs), I chose to work with tourist attractions only, excluding restaurants, cafes, and other similar establishments. This addition would be interesting to expand on in the future.

3.2 Data Collection

The dataset containing the touristic attractions in Romania was obtained using the *OpenTripMap API*[2]. To retrieve the data using the API, I needed to obtain center points for searching tourist attractions. To do this efficiently and ensure that the data is not lost, primarily due to the restrictions on the maximum number of locations returned in one query, I used data exported from *OpenStreetMap OverpassTurbo*[12] and cleaned it.

One method to create pairs of center points and radii would have been to take equally spaced points, in a grid-like manner, across the entire surface of Romania. This would not have been optimal because the distribution of the POIs is not uniform. It is evident that POIs tend to cluster in certain areas, such as large cities or tourist regions, and are sparse in the rest of the country. To find the pairs that would yield the best results, using the geometry data from the *Overpass Turbo* dataset. I calculated the centroids for each

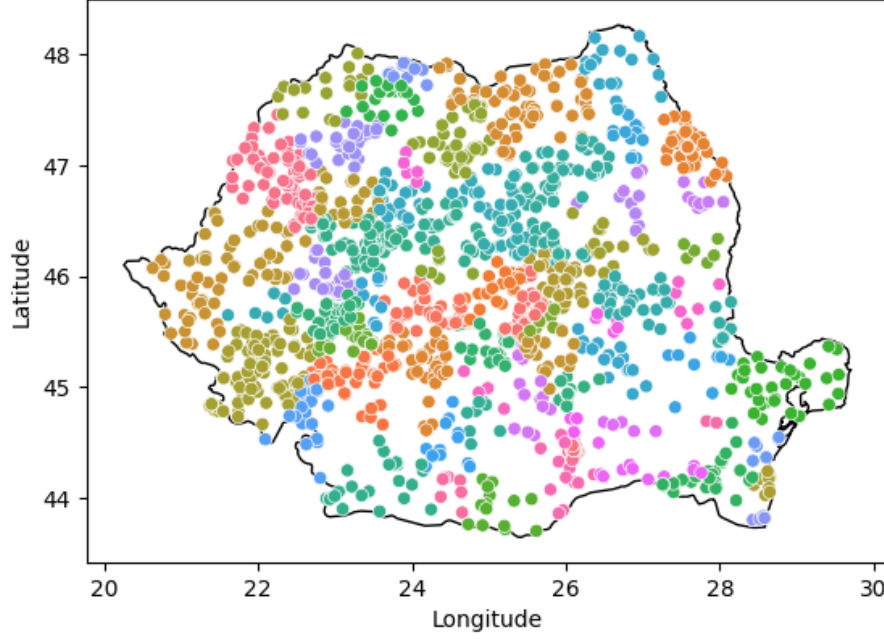


Figure 3.1: View of all Locations

geometry and used *BallTree* nearest neighbors algorithm from *scikit-learn* to create the pairs with which to obtain the data from the *APIs*.

With the help of the just created clusters, I get data from the two APIs, *OpenTripMap* and *Google Places*[7], but for the rest of the implementation, I will use the data from *OpenTripMap* and with the *Google Places API* will add some details that are used for the recommendation model and the scheduling algorithm. I have made this decision because the *Google API* yields fewer results and is less well-structured than the *OpenTripMap API* data.

Before making the predictions, the data must be cleaned to ensure that only high-quality data is processed and provided as output to the user.

3.3 Recommendation Model

3.3.1 Recommendation Model for a Single Individual

After collecting all the necessary data to make recommendations for each user preference, I developed three distinct recommendation algorithms.

Initially, I created a recommendation algorithm that filters all the locations based on proximity to the destination. Using a term frequency-inverse document frequency (TF-IDF) matrix and transformation, along with cosine similarity, the algorithm finds the

locations most similar to what the user wants. The final score for these locations is given by combining the cosine similarity with the rating and the rating counts from the Google details and the rate from the Open Trip Map details using this formula:

$$\begin{aligned} score = & 0.4 \cdot \text{Cosine Similarity score} + 0.2 \cdot \text{Google rating} + \\ & + 0.2 \cdot \text{OTM rating} + 0.2 \cdot \text{Google reviews count} \end{aligned} \quad (3.1)$$

For this formula to be correct, I scaled the three columns so the values range from 0 to 1. This algorithm does not yield the best results, as it is not tailored to the user’s profile.

Using a similar approach as in the paper Lewis et al. [10] for a second version of the recommendation algorithm, I implement a Retrieval-Augmented Generation system (RAG). The two main components of a RAG system are the retriever, which encodes the data and creates an index, and the generator component, which outputs the system’s response to the query. For the purpose of this implementation, the output is a score calculated by the large language model (LLM) based on the user’s preferences and the details of each point of interest (POI). With the help of the LLM, I also extract more information about each place, including whether the location is outside and the approximate visit time. This information will help with the generation of the optimal itinerary.

A third version of the recommendation algorithm uses a combination of the two approaches by adding the scores from the two previous algorithms. This was done to return places that are more ”popular” but still suitable for different user preferences.

3.3.2 Recommendation Model for a Group

The main challenge in making recommendations for multiple users is ensuring that each individual is considered when making predictions, while maintaining personalization, scalability, and fairness, particularly when preferences are contradictory. The most significant advantage of using an RAG system to make predictions is that the LLM has a better understanding of context. For example, if the group I want to make recommendations for consists of only two individuals who share a common interest in history, but one of them prefers museums. At the same time, the other enjoys visiting castles, fortresses, or archaeological sites. An algorithm similar to the first one I described earlier would not recognize that the two users have preferences that contradict each other, and would make recommendations based on popularity. An LLM would have the capacity to understand the context and make balanced recommendations.

Similar to the earlier approach, to determine the recommended places of interest for

a group, I implemented an algorithm that utilizes the RAG system described earlier, incorporating the complete user preferences for the entire group, and assigns a score, and a second algorithm that takes into consideration the Google ratings and the popularity of each location.

3.4 Scheduling Algorithm

The recommendation model has ranked the POIs. With this list of locations, I will create the itinerary that is optimized to minimize the distance traveled in a day and considers weather data.

For the weather data, I used the *Open-meteo API*[11] to obtain the weather prediction for the duration of the travel period, provided the trip ends within 16 days of the request being made. If the location is outside and it is raining, snowing, or the temperature is too low, the algorithm skips this location even if it was the next best stop. The skips are added back into the data of possible next steps to avoid removing important POIs for the entire trip period.

To determine the time and distance between two points, I used the *OpenRoute Service API*[6] for main transportation modes such as vehicle or walking. This data will be used in a distance matrix for the time and distance optimization. Before the itinerary is created, the algorithm requires certain inputs, such as the first and last days of the trip, the start and end times of each day, whether the user wants to take a lunch break, including the hours for it, and the destination of the trip.

The scheduling algorithm utilizes a function that returns the best next step with regards to a score that is calculated like so:

$$score = \frac{\text{recommendation score}}{1 + \frac{\text{duration}}{60}} \quad (3.2)$$

In this formula, the recommendation score is a number between 0 and 1, given by the recommendation algorithm, and the duration is the time (in minutes) it would take to travel from the current location to the next.

This function also checks if the location can be visited based on the weather and its opening hours, and if the time of the visit to the POI would fall during lunchtime. If it is during lunchtime, it simply delays the start time of the visit.

The itinerary creation function goes through each day in the trip period, from start to

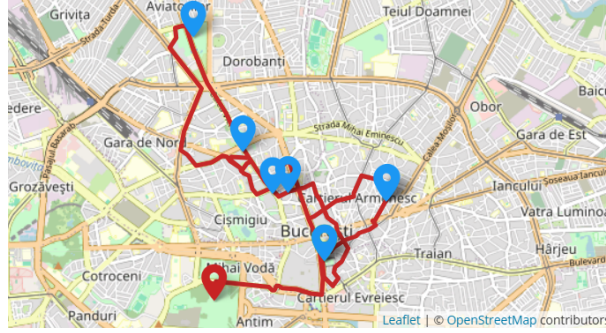


Figure 3.2: Example of recommended locations and route

finish. It creates an itinerary for each day by considering the next best location, the time required to complete the visit, and the time between the current location and the next best location, optimizing for minimal time spent transferring between the two locations. It ensures that a POI is visited only once, and if a location is skipped, it is added back to the list of possible next steps. The output of this function is a dataset containing the following columns: the name of the location, the approximate arrival and departure times for each POI, and the day of the trip. A route example is illustrated in Figure 3.2.¹

An interesting addition for this itinerary generator would be to recommend routes based on their attractiveness. This approach has been discussed in the paper Gu et al. [8] and could improve the user’s satisfaction with the overall recommended trip. Depending on each user’s personal preferences, they may prefer a more scenic route over one that gets them to their destination as quickly as possible, allowing for a more enjoyable vacation.

In the paper Erbil and Wörndl [3], a compelling method is proposed for creating itineraries based on the amount of time that can be filled with tasks or activities in a day. Adapting something similar in the future would reduce the compilation time of the itinerary generator (as shown in the paper) and also create better overall routes.

3.5 Interactive Interface

The interactive interface serves as the primary point of engagement between the user and the system, allowing users to input their preferences intuitively and easily view the results. This also contributes to the implementation’s user-friendly aspects. The interface was designed for casual users seeking quick access to travel recommendations and itinerary creation.

The interface was designed with ease of usability, responsiveness, and consistency in

¹This image was taken from the Open Route Service official API Playground with location coordinates taken from the system’s output from an example prompt

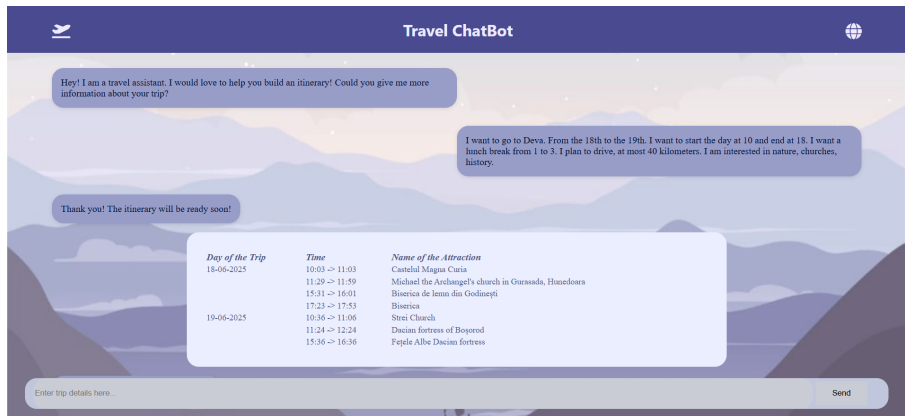


Figure 3.3: Example of conversation with the ChatBot through the interface

mind. The user can input their preferences for the trip they wish to have and how the itinerary should be created (such as destination and time intervals for the itinerary), and an LLM model asks leading questions to obtain all the missing data for the final result. ²

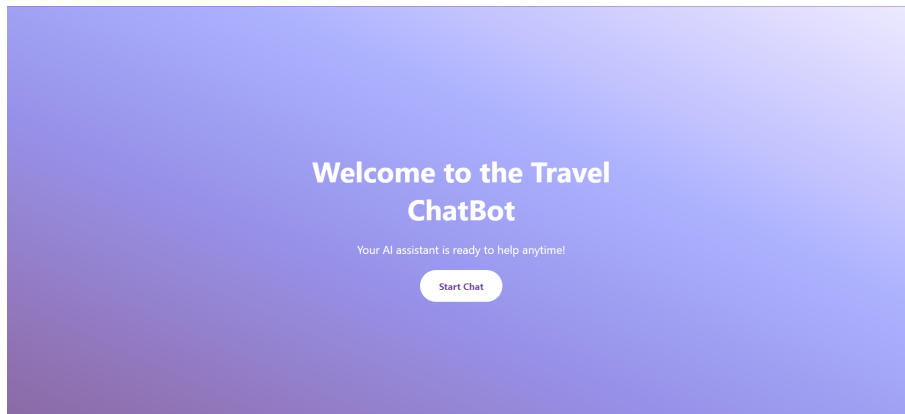


Figure 3.4: Landing page of the interface

²The background image for the chatbot conversation page is copyright-free and was taken from: https://www.freepik.com/free-vector/hand-drawn-flat-design-mountain-landscape_19964704.htm#fromView=image_search&page=1&position=0&uuid=7db25435-f504-4bf3-80ae-f54bf2ac4b37&query=Hand+drawn+flat+design+mountain+landscape+freepik

Chapter 4

Implementation

4.1 Overview

The purpose of this chapter is to provide further insight into the implementation methods chosen for each component as described in the previous chapter. The implementation involves combining multiple technologies and data sources, including external APIs, Python-based logic, and a conversational interface for user interaction.¹

4.2 System Architecture

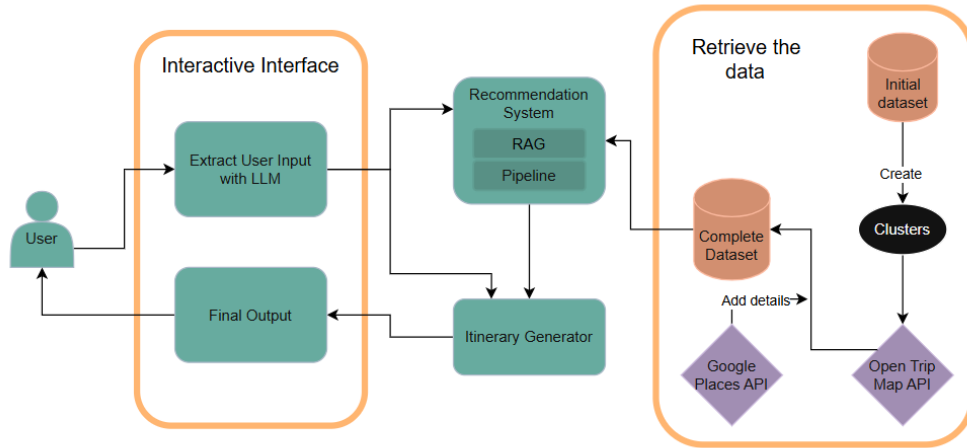


Figure 4.1: System Architecture

For ease of use and better understanding, I have structured the code into functions and Python files. The process has a total of four steps:

1. Collecting the data used for POI recommendations

¹The code is available at: https://1drv.ms/f/s!ArrAnLSZj9RTgUDSmsmEot_Svuwl?e=uiRI5d

2. Making POI recommendations with a RAG system based on the user inputs
3. Generating an itinerary with the optimal POIs based on the user inputs
4. Combining all the components into an interactive interface that extracts the user's inputs

4.3 Technology Stack

This project combines multiple technologies to support the development of a responsive, intelligent, and interactive travel recommendation system.

The frontend interface was developed using HTML, CSS, and JavaScript, creating an interactive environment that allows users to communicate with the chatbot. Asynchronous communication between the frontend and backend is handled using JavaScript's Fetch API. This ensures that the conversation proceeds smoothly. The backend component is built with Python, using the Flask framework, which handles routing, processes user input, and interacts with the large language model to generate relevant leading questions.

Natural language processing is performed using a GPT model, which takes the user's input text, extracts relevant information from it, and returns a JSON object containing the user's preferences for recommendation retrieval and itinerary generation. To retrieve tourist information, the system utilizes data from the OpenTripMap and Google Places APIs, leveraging the Requests library in Python. Data manipulation is implemented using Python libraries such as Pandas. The recommendations are retrieved from the full dataset and organized using an RAG system, which utilizes GPT models for both embeddings and chat completion. Additionally, geographical data for city center coordinates, administrative regions borders, and the geometry of Romania's borders were sourced from FlashWorldMap and SimpleMap.

This diverse technology stack enables the application to gather, process, and present personalized travel recommendations efficiently while remaining modular and extendable for potential future enhancements.

For all of the components to work well together, I have worked in a virtual environment. For reproducibility, I included the *requirements.txt* in the code files.

4.4 Implementation Details

4.4.1 Data Collection

This section describes the practical implementation of the system described in Chapter 3, specifically the data collection process. The implementation was done in Python, utilizing libraries such as *pandas*, *requests*, and *scikit-learn* for data manipulation, analysis, and API requests.

With the use of the dataset downloaded from *OverpassTurbo* and the *BallTree* algorithm from *scikit-learn*, I created clusters, represented by a center point and a radius, to retrieve all tourist attractions in Romania, as mentioned previously. This was the best approach for my specific task, because I was able to control both the distance metric (I used *haversine* to get a more accurate distance on the globe) and the approximate number of points contained in each cluster, as both the APIs I could be using for retrieving tourist attraction data have constraints on number of returned locations and range of search. The *OverpassTurbo* dataset does not contain the explicit latitude and longitude coordinates, but rather, the whole geometry of each location returned (the attraction outline, described through multiple points). To extract the approximate latitude and longitude, I used the centroid method from *geopandas* to determine the center of each geometry.

Using the newly formed clusters, I made requests to the *Open Trip Map API* for near search to obtain all the tourist attractions for each cluster. To ensure I retrieve only data relevant to the system’s purpose, I added constraints to the parameter lists specified in the API request. The parameter that filters locations based on their type is called *kinds*, and I only kept the POIs related to culture, history, architecture, natural, amusement, skiing, climbing, pools, and other winter sports. I also filter based on the popularity of the location, kept in the parameter *rate*. I perform this last step because I have observed that data with a rate of 1 (the lowest) tends to be noisy and includes irrelevant locations.

Even with the previous filtering, some irrelevant locations were retrieved, so I performed additional filtering based on type, removing hospitals and religious attractions (which I later reinstated, but only for the most popular ones), as well as other types.

4.4.2 Data Processing

The dataset from *OpenTripMap* is better suited for this task than the *Google Places* dataset, although it requires several processing steps to be utilized in the remainder of the implementation. The first step is to remove all rows that have a *null* value in the *name* column and also to remove all locations outside the borders of Romania. The clusters

previously created are, unfortunately, also generating some results from the neighboring countries. Using a dataset that contains the geography of Romania[5], the locations that are not within the borders are filtered out.

To get additional location data for my dataset - for now the only location data is in the columns latitude and longitude - I will add three more columns: administrative region, using the geography of each administrative area from *FlashWorldMap*[4], closest city, using the city center points data from *SimpleMaps*[15] and another column which holds the distance between the city center and each location.

To obtain additional data that will aid in developing the recommendation model, I utilize the *Google Places API*. From this, with text based search by the name of the location with a proximity constraint (already integrated in the text search from the API), I extract the rating, review count, written reviews (limited to 5 for each location), opening schedule, and other relevant information (I also extracted the price range but it was missing for most locations so an optimization or filter based on this column is unavailable).

4.4.3 Recommendation System

To implement the recommendation system described in Chapter 3, a Retrieval-Augmented Generation (RAG) system is employed to enhance the accuracy and context-awareness of the recommendation scores generated by the large language model (LLM). The RAG system consists of four components: query encoder, retriever, document corpus, and generator.

The entire tourist attractions dataset is filtered to contain only the relevant data for making recommendations (such as name, Wikipedia description, reviews, etc.). This dataset is encoded using the *text-embedding-3-small* model from the *OpenAI API*, and the corpus is saved as a Parquet file for later use.

The functions that return the top-k recommendations are called *getTopRecommendation* and are divided into four groups: if the recommendation is made for an individual or a group, and if the score is calculated purely by the LLM or a hybrid model. These functions take as inputs the dataframe used, the coordinates of the starting point, the maximum distance the user is willing to travel between the starting point and a location, the number of days of the trip, the maximum number of activities per day, and the user's profile. The first step is to calculate the necessary k locations to return. This is done by taking into consideration the number of days, to which I added one for some padding, and the maximum activities per day. From the entire dataset, I filter so that only loca-

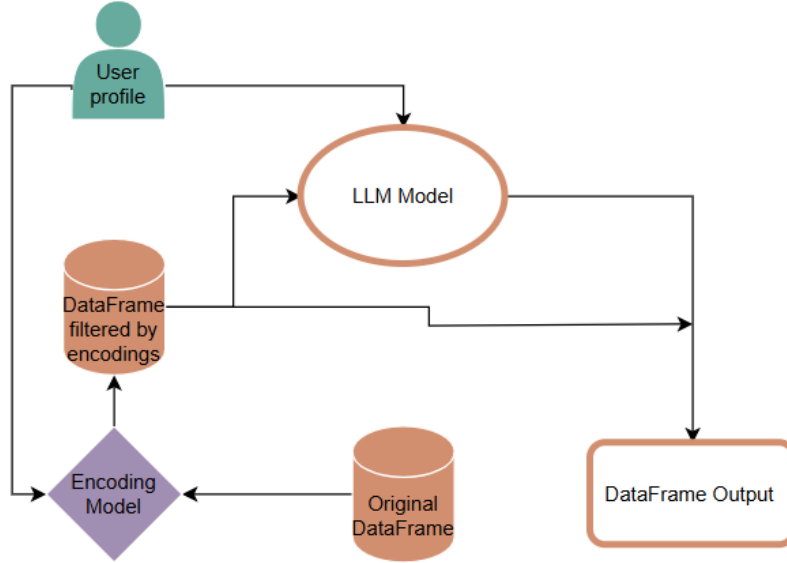


Figure 4.2: Recommendation System Architecture

tions within the maximum distance radius are considered, and by taking the indexes from this filtered dataset, I also filter the embeddings dataset. With the filtered embeddings dataset, I create the index using *FAISS*. The next step is to encode the query, in this case, the user's profile, with the same embedding model from earlier. With this encoded query and the index, I search for the top-k locations based on the similarity between embeddings.

The top-k locations just received are given to the large language model one by one, along with the user's profile, for the final score output. The LLM is then instructed to assign a score to the location, ranging from 0 to 100, based on its compatibility with the user's preferences, whether the location is outdoors or not, and the approximate time required for a complete visit to the location. This data is added to the filtered dataset and then returned. If this function only considers the score calculated by the LLM, it stops here. However, if the hybrid score is required, the mean between the LLM score and the rating score is returned. The rating score's formula is described in Chapter 3 in 3.1.

4.4.4 Itinerary Generation

After the recommended locations have been retrieved and the user has given enough information about how they wish their trip to be structured, the system creates and outputs the itinerary.

The function that returns the finished itinerary is called *getSchedule*. This function takes as parameters the recommended locations, the trip preferences (the start and final

days of the trip, the start and final hours of activities for each day, if the user wants a lunch break, and if so, the time interval for it, the starting point's latitude and longitude, the principal mode of transportation between two locations, and the maximum number of activities done in a day) and returns, by days of the trip, the time intervals for each activity and the activity name.

Firstly, the distance matrix is obtained using the coordinates of each location and the mode of transport(car or walking) with the help of the *Open Route Service API*[14]. If the user wanted to take public transport between the locations, a solution would have been to utilize the *Open Transport API*. However, this only provides the duration between two locations (there is no distance matrix API), and using it would take a long time compared to the other possible choices, so this option is not implemented. From the recommended locations dataset, I extract only the relevant columns (those no longer needed for making recommendations), such as coordinates, the schedule for each location, and the recommendation score. I also need weather data for the trip duration (to make sure the itinerary is realistic), which is obtained using *Open Meteo API* for a period of 16 days(from the day the request is made), taking the data about temperature and possible rain, showers, or snowfall. The first location, the starting point, is added to the list of visited locations (which was previously empty) to ensure that each day begins at the same point.

After retrieving the initial data using two while loops, the algorithm proceeds through each day and each itinerary hour, searching for the best next location. A location is considered "next best" if it has the highest score of recommendation in relation to distance (as explained in Chapter 3) and if a visit to that location is possible. A visit is possible if the location is open at the time the activity would be finished, and if the location is outdoors, the weather is favorable (with no extreme temperatures or precipitation, such as rain or snowfall). If the visit is not possible, the location is appended to a separate list. It will be added later to ensure that every possible location is considered at each available time. After the best next location is added, its ID is added to the list of visited locations to make sure no location is visited twice. After the while loops end, the itinerary is formatted and returned for the user to view.

4.4.5 Interactive Interface

The interactive interface was created using the *Flask* library in Python, an *HTML* template, the style *CSS* file, and the *JavaScript* file for interactivity and animations.

Using a text input field in HTML with a "Send" button, the user can send messages

in natural language to the large language model, which is tasked with extracting relevant information from the user’s messages about the trip. This is made possible with a *POST* request method in the Python file and a *fetch()* method in the JavaScript file, which extracts the text input from the HTML and passes it to the request in JSON format. The LLM used to extract information from the input is *gpt-4o-mini-2024-07-18* from the *OpenAI API*, which requires an API key. I previously attempted to use a local model, *mistral-7b-instruct-v0.1.Q4_K_M*, but the computational time was too long and the results were not satisfactory.

The information is extracted by providing the LLM with the user input text and a JSON object containing missing values to be filled. If one of the values is still missing, the LLM asks the user a leading question. The question is visible in the chatbot web interface by sending the text to be appended in the HTML with a JavaScript function. Once all the necessary values for generating the recommended locations are filled, the algorithm computes the recommendations. After all the values are filled, the itinerary is generated and appended to the HTML through a JSON HTTP response. JavaScript uses the *fetch* method to get the data and sends it to the HTML through a function that appends a new child to the body of the file. The itinerary is formatted as a table with the day of the trip, time interval, and name of the attractions as columns, and each attraction is displayed on a separate row.

The next step after the itinerary is presented is to ensure that the user is satisfied with the result, or if they would like to make any changes. After the user says they are happy with the itinerary, the chatbot outputs a simple final message.

I also implemented a pocket version of the system. This pocket version is compatible with locations in Bucharest and features five categories of possible interests: cultural, art, history, sports, and outdoor activities. This version was created for a quicker return of the itinerary, primarily for showcasing purposes.

4.4.6 Output Example

For a better understanding of the system outputs and interface interaction, I will present a complete example here. The conversation with the chatbot was:

- LLM Hey! I am a travel assistant. I would love to help you build an itinerary! Could you give me more information about your trip?
- User I wish to go on a trip to Bucharest from the 19th to the 20th. I want the day to start at 10 and end 8 hours later. I would like a two-hour break starting at 1.

- LLM What is your preferred mode of transport: car or walking?
- User I prefer to use the car. I am willing to drive up to 10 kilometers.
- LLM What is your group size?
- User Only one person.
- LLM What are your interests?
- User I like art and museums. I also enjoy walks in the park.
- LLM Thank you! The itinerary will be ready soon!
- LLM *insert itinerary*
- LLM Would you like to make any changes?
- User No, I am done!
- LLM Done! Your trip plan is complete

Name	Score	Outdoors	Time Spent
National Museum of Contemporary Art	0.8	no	2 hours
Parcul Kiseleff	0.9	yes	0.5 hours
Theodor Aman Museum	0.8	no	2 hours
Theodor Pallady Museum	0.8	no	2 hours
National Museum of Art of Romania	0.9	no	2 hours
Art Collections Museum	0.9	no	2 hours
Romanian Kitsch Museum	0.7	no	2 hours
Ditarie Gusti National Village Museum	0.9	yes	2 hours
Rotonda Scriitorilor	0.8	yes	0.5 hours
Carol Park Mausoleum, Bucharest	0.8	yes	1.5 hours

Table 4.1: Table of Recommendations

For those locations, the itinerary is:

Day of the Trip	Time	Name of Attraction
19-06-2025	10:11 -> 12:11	National Museum of Art of Romania
	15:02 -> 17:02	Art Collections Museum
	17:09 -> 17:39	Parcul Kiseleff
20-06-2025	10:10-> 12:10	Ditarie Gusti National Village Museum
	15:08 -> 17:08	Theodor Aman Museum
	17:19 -> 17:49	Rotonda Scriitorilor

Table 4.2: Itinerary

Chapter 5

Results and Evaluation

This chapter presents the results produced by the travel recommendation system and evaluates its performance based on criteria such as relevance, personalization, and user preference.

5.1 Information Extraction

An essential component of the system involves extracting relevant information about the user's preferences to create the itinerary and recommend points of interest (POIs). This includes details such as destination, the date of the trip, the times the itinerary should be made, whether to include a lunch break, and so on. I explained the extraction process using an LLM in Chapter 4. Some examples of information extraction are presented in Table 5.1.

User natural language input	Extracted Information
I want to travel to Bucharest from the 15th to the 17th. I like museums and art.	Destination: Bucharest Start Day: 15-06-2025 Number of Days: 3 Final Day: 17-06-2025 Interests: museums, art
Can I change the destination to Iasi?	Destination: Iasi
The day should begin at 10 and end eight hours later. Add a break from 1 that lasts two hours.	Start Hour: 10:00 Duration: 08:00 Final Hour: 18:00 Lunch: True Lunch Start: 13:00 Lunch Duration: 2 Lunch End: 15:00

I enjoy sports. I want to use the car to get from one location to another. I also enjoy activities that take place outside.	Interests: sports, outdoor activities Transport: car
I am traveling with a group of people. One of them enjoys cultural activities while the other likes sports and arts. I am particularly fond of history and religious sites.	Interests: [[cultural activities], [sports, arts], [history, religious sites]] Group: True

Table 5.1: Example of user preference extraction

It is essential to note that, for certain user inputs, the LLM automatically assigns a false value to the Lunch field. This can be changed if the user asks for it, but the LLM does not ask direct questions about a possible lunch break in the itinerary.

Apart from the previously mentioned error, the LLM assigns the correct values and asks relevant follow-up questions.

5.2 System Evaluation

5.2.1 Point System

This evaluation will be conducted by assigning a score from 0 to 5 to the results of different prompts. A point will be given for any of the following that are true: there are no hallucinations(No hallucinations), the recommendations are relevant to the user’s preferences(Relevancy), interesting suggestions (not just the most popular places)(Novelty), the prompt is respected(Prompt alignment), and no answers to unrelated tasks(Task focus). For ease, I have entered complete prompts in the chatbot, and if it asked another question, I responded with "I do not care" or "Done."

Prompt	Evaluation	RAG	Hybrid	Best
I plan to travel to Cluj from the 16th to the 17th. I want to walk between locations, at most 3 kilometers. The day should start at 11 and end at 7. I will do at most four activities per day. I enjoy the outdoors, parks, and art.	No hallucinations	1p	1p	RAG
	Relevancy	1p	1p	
	Novelty	0.5p	1p	
	Prompt alignment	1p	0.5p	
	Task focus	1p	0.5p	
	Total	4.5p	4p	
I want to travel to Bucharest. From the 18th for 2 days. I like art museums and history. One friend also likes history and parks. Our other friend likes art and outdoor activities. The day should start at 12 and end 7 hours later. We do not want a break. We will use the car and drive at most 20 kilometers.	No hallucinations	1p	1p	RAG
	Relevancy	1p	0.5p	
	Novelty	0.5p	0.5p	
	Prompt alignment	1p	0p	
	Task focus	1p	0.5p	
	Total	4.5p	2.5p	
I want to travel to Ploiesti. I wish to only stay one day, on the 19th. I like history and science. I want the day to start at nine and a half and end 10 hours later. A lunch break from 2, that lasts an hour and a half, would be ideal. I plan to walk, at most 3 km.	No hallucinations	1p	0.5p	RAG
	Relevancy	0.5p	0.5p	
	Novelty	0p	0p	
	Prompt alignment	1p	1p	
	Task focus	1p	0.5p	
	Total	4p	2.5p	

Overall, the RAG system yields the best results, as it is better equipped to understand context. However, depending on the task, the hybrid system is similar and sometimes yields results that are more diverse than those of the RAG system.

5.2.2 User Survey

To test how the interactive platform, recommendations, and itinerary would be perceived in the real world by the public, I conducted a user survey with a small sample of users. The questions used for the survey are:

From the survey, I can conclude that the interface could be visually improved; most respondents stated that they found the colors a bit too dull. The speed was considered good, but for some prompts, it took longer; therefore, an improvement in this aspect is

Use	Have you used a chatbot before?
Clarity	Was anything confusing or unclear?
	What was unclear?
Visual	How would you rate the visual design?
Behavior	Did the system behave as you expected?
Satisf.	Did you find the recommendations to your satisfaction?
Speed	Did the system respond quickly enough?
Best	Which answers did you prefer?

Table 5.2: Survey Questions

Table 5.3: Comparison of RAG and Hybrid recommendations and itinerary

User	Use	Visual	RAG		Hybrid		Clarity	Best
			Behavior	Satisf.	Behavior	Satisf.		
User 1	no	4	4	4	5	5	4	Hybrid
User 2	no	4	5	5	4	5	5	Tie
User 3	yes	5	5	3	5	4	4	Hybrid
User 4	yes	4	5	5	5	5	5	Hybrid
User 5	yes	3	5	4	5	3	5	RAG

necessary. An element that was generally considered confusing until I explained it was the *maximum distance* question or field, with respondents asking if this meant the total distance or maximum distance between two locations, but understanding once I explained (It would be more intuitive to be the total maximum distance). Overall, most respondents found recommendations that they liked from both models, but their favorite tends to be the Hybrid model. Depending on how they expressed their interests, one of the models seemed to perform better than the other. For example, if one of the interests was *botanical gardens*, if there were no such places in the selected area, the RAG system still provided some outdoor activities, such as parks, while the Hybrid model concentrated on the other interests from the user inputs.

Chapter 6

Discussion

6.1 Strengths and Limitations of the Implementation

The implementation of the travel recommendation system presents several strengths that contribute to its effectiveness and usability. One of the main strengths lies in its ability to personalize itineraries based on user-defined preferences such as group size, means of transportation, and areas of interest. This level of personalization enables the system to produce relevant plans. Another advantage is the automation of the entire itinerary planning process, from natural language input to structured recommendations. This reduces the need for manual intervention, offering a seamless user experience. Furthermore, the integration of multiple APIs, such as OpenTripMap and Google Places, enhances the richness and accuracy of the data used for recommendations. The system's modular structure also makes it easier to maintain and expand in future iterations.

However, the system also has limitations. One major limitation is the quality and coverage of data, as some APIs may provide incomplete or outdated information, particularly in less popular areas. Additionally, while the use of a language model significantly enhances user interaction, it may occasionally misinterpret user input or inconsistently populate structured fields. Another constraint is scalability — the current implementation may face performance issues if scaled to support a large number of users simultaneously. Lastly, the system has not undergone extensive real-world testing, which is crucial to evaluate its robustness in diverse and unpredictable user scenarios.

6.2 Implications and Applications

This work has promising implications and practical applications in the domain of travel planning and recommendation systems. By enabling fast and personalized itinerary generation, the system can greatly benefit casual travelers who seek quick and tailored travel

plans without needing to conduct extensive research. Travel agencies and tourism platforms could also integrate such a tool to automate trip planning and enhance customer service.

Academically, it contributes to the intersection of natural language processing using an LLM and a recommendation system, demonstrating how natural language processing can be combined with geographic and user-preference data to facilitate intelligent decision-making. The chatbot interface enhances accessibility, enabling non-technical users to interact easily with complex systems using everyday language.

Chapter 7

Future Work and Conclusion

Several improvements can be explored to enhance the system further. Integrating real-time traffic updates or live events could make the recommendations more responsive and appropriate. Expanding the location dataset to include additional countries would increase the system’s applicability. Another area of development worth mentioning is the itinerary generator, which could be optimized to reduce computational time. The large language model used could also be fine-tuned to enhance the accuracy of interpreting user input and making recommendations. Finally, deploying the application as a full web or mobile service, with the option to track user-visited locations and access previous reviews of locations, could lead to more accurate recommendations in the future.

This dissertation paper presents the development of a travel recommendation system designed to generate personalized itineraries based on user preferences and travel constraints. By combining a chatbot interactive interface with the system, the program is capable of providing users with an itinerary tailored to them.

Bibliography

- [1] Ashmi Banerjee, Adithi Satish, and Wolfgang Wörndl. “Enhancing Tourism recommender systems for sustainable city trips using Retrieval-Augmented Generation.” In: *Springer Communications in Computer and Information Science*, vol 2470, 2025 2470 (Jan. 2025), pp. 19–34. DOI: [10.1007/978-3-031-87654-7_3](https://doi.org/10.1007/978-3-031-87654-7_3). URL: https://doi.org/10.1007/978-3-031-87654-7_3.
- [2] OSM contributors. *OpenTripMap API*. Accessed : 14.05.2025. URL: <https://opentripmap.com/en/#7/45.993/25.164>.
- [3] Elif Erbil and Wolfgang Wörndl. *Generating Multi-Day round trip itineraries for tourists*. Vol. 1. 2021. URL: https://ceur-ws.org/Vol-2855/main_short_1.pdf.
- [4] Formerly FlashUSAmapi.com and FlashWorldMap.com. *Admin Aread (level 1)*. Accessed : 2010-2025. URL: <https://simplemaps.com/gis/country/ro>.
- [5] Formerly FlashUSAmapi.com and FlashWorldMap.com. *All of Romania*. Accessed : 2010-2025. URL: <https://simplemaps.com/gis/country/ro#all>.
- [6] HeiGIT gGmbH. *openroute service API*. Accessed : 2024. URL: <https://openrouteservice.org/>.
- [7] Google. *Google Places API Legacy*. 2024. URL: <https://developers.google.com/maps/documentation/places/web-service/overview>.
- [8] Jiqing Gu, Chao Song, Wenjun Jiang, Xiaomin Wang, and Ming Liu. “Enhancing Personalized Trip Recommendation with Attractive Routes.” In: *Proceedings of the AAAI Conference on Artificial Intelligence* 34.01 (Apr. 2020), pp. 662–669. DOI: [10.1609/aaai.v34i01.5407](https://doi.org/10.1609/aaai.v34i01.5407). URL: <https://doi.org/10.1609/aaai.v34i01.5407>.
- [9] Sajal Halder, Kwan Hui Lim, Jeffrey Chan, and Xiuzhen Zhang. “A survey on personalized itinerary recommendation: From optimisation to deep learning.” In: *Applied Soft Computing* 152 (Dec. 2023), p. 111200. DOI: [10.1016/j.asoc.2023.111200](https://doi.org/10.1016/j.asoc.2023.111200). URL: <https://doi.org/10.1016/j.asoc.2023.111200>.

- [10] Patrick S. H. Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-Tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. “Retrieval-Augmented Generation for Knowledge-Intensive NLP tasks.” In: *Neural Information Processing Systems* 33 (May 2020), pp. 9459–9474. URL: <https://proceedings.neurips.cc/paper/2020/file/6b493230205f780e1bc26945df7481e5-Paper.pdf>.
- [11] Open-Meteo. *Open-meteo API*. Accessed : 2022-2025. URL: <https://open-meteo.com/>.
- [12] OpenStreetMap contributors. *Locations dump retrieved from*. Accessed : 23.02.2025. 2017. URL: <https://www.openstreetmap.org/>.
- [13] Jinhu Qi, Shuai Yan, Yibo Zhang, Wentao Zhang, Rong Jin, Yuwei Hu, and Ke Wang. “RAG-Optimized Tibetan Tourism LLMs: enhancing accuracy and Personalization.” In: *arXiv (Cornell University)* (Aug. 2024). DOI: [10.48550/arxiv.2408.12003](https://doi.org/10.48550/arxiv.2408.12003). URL: <http://arxiv.org/abs/2408.12003>.
- [14] Open Transport Routing. *Routing API*. Accessed : 2025. URL: <https://opentransport.ro/en/developers/apis/1-routing-api/>.
- [15] SimpleMaps.com. *Romania Cities Database*. Accessed : 2010-2025. URL: <https://simplemaps.com/data/ro-cities>.