

Using Natural Language Processing to interact with geospatial data

By: Valentijn Embrechts

Supervisors: Andreas Kamilaris and Asfa Jamil

Critical Observer: Ozlem Durmaz

Creative Technology Bachelor Graduation Project

18-7-2024



CYENS
CENTRE OF EXCELLENCE

Abstract

Geospatial data can provide valuable insights into environmental patterns, (urban) development and resource management. Currently the analysis and interpretation of geospatial data requires knowledge of complex softwares and methods, effectively limiting the power of geospatial science to a small group of knowledgeable individuals. Background research shows a lack of current end to end natural language systems that perform question answering while referencing geospatial data. Using the Creative Technology design process and co-design, multiple natural language question answering systems were prototyped, during specification one system was chosen to be built for production. In a constant loop of evaluation and optimization individual modules were improved. The system met the requirements of being speedy, accurate and non-proprietary / self hosted. The tested speed was under 10 seconds on average, the system accurately answered questions 92% of the time, struggling more with complex questions than simple ones. Lastly the system can be run on a local machine and does not depend on proprietary software. This new geospatial question answering system allows inexperienced users to interact with and use geospatial data to improve our world.

Acknowledgements

I would like to sincerely thank everyone who supported and guided me throughout this project. Firstly my deepest thanks go to my project supervisor, Andreas Kamilaris, whose guidance and feedback played an important role in my project and my own personal development. I also would like to thank my critical observer, Ozlem Durmaz, for being critical, asking pertinent questions and giving me constructive feedback.

Besides my supervisor, other members of the SuPerWorld team have provided invaluable support. Asfa Jamil, an expert in many different forms of artificial intelligence guided me through the complex world of machine learning and large language models. Aytac Guley, a fullstack developer at SuPerWorld worked closely with me in the final realization steps of this project. Aytac helped me realize a program into a working api that has been integrated into Gaea.

Additionally I used resources from the Utwente (jupyter) cluster and would like to thank the University of Twente and the staff maintaining the cluster. Without the computational capabilities this project would have taken much longer. I would also like to thank the creators and maintainers of the open source tools and systems that I used, such as Spacy, Langchain and LLama3. Without their neatly written and structured documentation, this project would not have been possible.

Finally I would like to express gratitude to my friends and family, who at tough times provided me with motivation and support, they also provided me with critical feedback that allowed me to improve my work.

Table of Contents

Abstract	2
Acknowledgements	3
Table of Contents	4
List of figures	6
Chapter 1 - Introduction	9
1.1 Introduction	9
1.2 Research Questions	10
Chapter 2 - Background Research	11
2.1 Literature research	11
2.1.1 NLP in Geospatial Sciences	11
2.1.2 Possible use of LLMs in Geospatial Sciences	13
2.2 State of the art	14
2.3 Reflection on background research	15
Chapter 3 - Methods and Techniques	16
3.1 Co-design	16
3.1.1 Engaging & Understanding	16
3.1.2 Ideation & Validation.	16
3.2 CreaTe Design Process	17
3.2.1 Ideation	19
3.2.2 Specification	19
3.2.3 Realization	19
3.2.4 Evaluation	20
Chapter 4 - Ideation	21
4.1 Basic NLP System	21
4.2 Endpoint Recognition	22
4.3 Answer Generation	22
4.4 Complexities	22
4.5 Autonomous API Calling	23
Chapter 5 - Specification	24
5.1 Functional Requirements	24
5.2 Non-functional requirements	24
5.3 Autonomous LLM vs. NLP system	24
5.4 System Selection	25
Chapter 6 - Realization	27
6.1 Endpoint prediction	29
6.2 Location extraction	29
6.3 Conversion of location to coordinates	30
6.4 Requesting data from the GAEA API	31
6.5 Generating a natural language answer using an LLM	32

6.6 Training the endpoint prediction module	33
6.7 Training location recognition model.	37
Chapter 7 - Optimization	42
7.1 Endpoint extraction	42
7.2 Extract location	43
7.3 Converting the location to coordinates	47
7.4 Requesting data from the Gaea GeoApi	48
7.5 Generating a natural language answer	49
7.5.1 Prompt Structure 1	50
7.5.2 Prompt Structure 2	50
7.5.3 Prompt Structure 3	51
7.5.4 Prompt Structure 4	51
7.5.5 Comparing prompt structure	52
7.6 API Wrapper & Chatpage	53
Chapter 8 - Evaluation	56
8.1 Answer evaluation	56
8.2 Requirements evaluation	56
8.1 Accuracy	56
8.2 Speed	58
8.3 Local Deployment	59
8.4 Integratability	59
Chapter 9 - Discussion & Future work	60
Chapter 10 - Conclusion	61
References	62
Appendix A - Template questions	65
Appendix A1 - Vicinity related Template Questions	65
Appendix A2 - Questions related to weather	66
Appendix A3 - Weather related template questions	69
Appendix A4 - Template questions relating to vicinity	70
Appendix B - API Documentation	71
/query endpoint	71
/rate_response endpoint	72
Appendix C - Evaluation questions	74

List of figures

Figure 5.4.1: Results of comparing autonomous LLM V1 (blue), autonomous tool LLM V2 (red) and the NLP system (yellow) to each other.	27
Figure 6.1: NLP system pipeline overview from question to final answer, showing the five intermediary steps.	28
Figure 6.2: Global overview of how data is passed between the different modules and how modules interact with each other.	29
Figure 6.1.1: inference code for using the fine tuned Bert classification model to predict the endpoint relevant to that question. (Boilerplate code that was not deemed important for explanation has been left out, it can all be found in the Meliferea git repository [21]).	30
Figure 6.2.1: Using the 'model-best' that was trained the locations in questions are identified and visually shown using displacy (figure 6.2.2).	30
Figure 6.2.2: Shows the output of the code in figure 6.2.1.	31
Figure 6.2.3: Logic used to extract a list of locations from the question.	31
Figure 6.3.1: Code that converts the locations into coordinates which is an excerpt out of the main run() code from figure 6.2.	32
Figure 6.3.2: Using the locationIQ geocoding api the location name is converted into a tuple containing the latitude and longitude.	32
Figure 6.4.1: Excerpt from figure 6.2 that loops through all the coordinates and makes a request for each coordinate to get the relevant data then adds that data to the list.	33
Figure 6.4.2: Request data function that handles API requests to the GeoAPI and returns the json data.	34
Figure 6.5.1: Excerpt from figure 6.2 that passes the data from the Gaea GeoAPI and the original question into the process_answer_data function.	34
Figure 6.5.2: Inside the process_and_answer function the llm is started and then for every call it is prompted to use the data and original question to generate a natural language answer.	35
Figure 6.6.1 : Datasets for each category of endpoints are imported (due to the length of the dataset these have been left out of this report, but can be found at the github repository [21]). These datasets are merged together and split into two lists, the first containing the questions and the second containing the endpoints.	35
Figure 6.6.2: To simply classification the different possible endpoints are mapped to an enumerator. Each endpoint in the list is then replaced with its corresponding number.	36
Figure 6.6.3: The dataset is then split into a training dataset and a validation dataset. 80% of the questions are for training and 20% are for validation. A fixed random state was used to allow for easy reproduction of the results.	37
Figure 6.6.4: Training and evaluation loop. (hidden: conversion of the dataset into the dataloader done before training)	38
Figure 6.6.5: Lastly the model needs to be saved, this is done in the models/endpoint folder. The model name is dependent on the amount of epochs and the last accuracy. This is for easy identification later.	38
Figure 6.7.1: Code that imports the csv's that contain the questions and items that are substituted in. Finally questions and annotations are added to their respective lists. (hidden:	

conversion from dictionary to lists and definition of lists).	39
Figure 6.7.2: Converting the data in the lists to a format that spacy can use to fit the model.	40
Figure 6.7.3: Converting each entry into an annotated Spacy doc. These are added to a docbin which is saved for training	40
Figure 6.7.4: Using the training config file (config.cfg) and the training files to train the NER model.	41
Figure 6.7.6: Output of the code in figure 6.7.5, shows the location and topic that the system identified in the question.	42
Figure 7.1.1: Different datasets that GAEA offers to users. From A. Jamil et al [1]	43
Figure 7.2.1: Results from testing three fine tuned models on four different locational formats. Average accuracy per model is also shown	44
Figure 7.2.2: Location detection model class that (down)loads the model on initiation. The text can be passed into the extract function to extract a list of different locations.	45
Figure 7.2.3: Programmatically combine all locational names together when nothing is found using the coordinate search.	46
Figure 7.2.4: When we still can't find a location then try splitting the combined string from figure 7.2.3 in two and try again.	46
Figure 7.2.5: When both logic approaches shown in figure 7.2.3 and figure 7.2.4 don't yield any coordinates for locations in Cyprus we notify the user and advise them to try capitalizing locational pronouns.	47
Figure 7.3.1: Programmatically convert a location name into coordinates by first checking the caching database, else making a request to the external geocoding API.	48
Figure 7.4.1: If the api is down and unusable, the system always returns an error stating that it cannot get any information. This is done to prevent hallucination by the LLM due to missing data.	49
Figure 7.4.2: Explaining what the data from the API is on by mentioning the endpoint that it is from and the location for which it is, after which it is added to the data list.	49
Figure 7.4.3: User defined custom data filtering rules implemented to prevent and overload on data for the answering LLM.	50
Figure 7.5.1: A simple prompt explaining the context of geospatial answer generation with the user question and API response data plugged in.	50
Figure 7.5.2: Using conversational chat history to split the instructions from the user question and data that is to be used.	51
Figure 7.5.3: Use a conversational chat history to pass a contextual system message, user question and the data needed to answer the question.	52
Figure 7.5.4: Using human and system messages to pass all the data into the LLM for answering. Using more human messages to reinforce the importance of using the API response data for answer generation.	53
Figure 7.5.5: Results of testing the different answer generation approaches shown in figure 7.5.1 up to figure 7.5.4. The code in figure 7.5.1 is shown as structure 1 (blue), the code in figure 7.5.2 is shown as structure 2 (red), the code in figure 7.5.3 is shown as structure 3 (yellow) and lastly the code in figure 7.5.4 is shown as structure 4 (green).	53
Figure 7.6.1: Using Flask to wrap the run function, allowing the system to accept rest API requests (full code can be found on github [21]).	54
Figure 7.6.2: A simple web app that interacts with the production API and displays answers in a text box, green if the API returns success as true and red if success is false. Users can also	

easy rate answers using the thumbs up and thumbs down.	55
Figure 8.1.1: Question answering text box where a user can rate their question using a thumbs up or thumbs down.	56
Figure 8.2: System accuracy per complexity level for the questions defined in appendix C.	57
Figure 8.2: Comparing the production system (GeoAI) accuracy to the prototype system (NLP, LLM v1, LLM v2).	58

Chapter 1 - Introduction

1.1 Introduction

By using sensors that monitor specific aspects of our earth, we can learn more about a geographic region. In earth observation we use sensors on planes, buoys and satellites to collect a vast amount of data over a large spatial region. Using this data we can identify spatial trends such as how a certain area like a delta is different from its surroundings or temporal trends like average temperature of an area rising over the span of decades, which is also known as global warming. Besides analysis of historic trends we can also use this data for future modeling to better understand the processes of our earth and make more well informed decisions.

Traditionally accessing and manipulating geospatial data has required knowledge of complex geographic information systems (GIS). The steep learning curve associated with GIS software creates a significant barrier which limits the ability of non-experts to utilize this data to its full potential. Lowering this barrier would allow more people to use geospatial data to make better informed decisions and allocate resources more efficiently and effectively.

Besides collecting data the SuPerWorld team aims to democratize access with their innovative online tool to allow everyone to access and understand geospatial data. Despite the efforts of the SuPerWorld team and their GAEA webtool there still appears to be a need for a simple tool that does not require any understanding of GIS tools to get simple data points [1].

Natural language might provide a solution to the inaccessibility of geospatial data, a type of digital chatbot that could reference geospatial data. This could fundamentally change the way we interact with geospatial data. It would allow inexperienced users to ask the system questions just like we would ask a friend or a colleague. It would provide quick answers and remove the need for complex GIS software for quick and simple queries, thus allowing anyone with an internet connection to use the power of geospatial data to improve our world.

1.2 Research Questions

The main research question is

How can a geospatial question answering system be built?

The following sub-questions are formed to answer the research question:

1. How is natural language processing currently being used in geospatial sciences?
2. How can a system that uses natural language processing to interact with geospatial data be realized?
3. How could large language models be used to interact with geospatial data?

Chapter 2 - Background Research

2.1 Literature research

2.1.1 NLP in Geospatial Sciences

Natural language processing (NLP) is a machine learning technology that allows computer systems to understand the meaning of words and the relationship between words in texts. Primitive historic NLP systems relied on user defined rules, whereas modern systems use complex statistical methods to determine the meaning of words. McKenzie and Adams [2] have identified three main applications of natural language processing in geospatial sciences. Firstly toponym disambiguation, secondly identification of spatial relationships in text and lastly identification of thematic patterns. These three uses also each come with their own challenges respectively. We must solve the respective problems before being able to use NLP to its full potential for geospatial sciences.

Thematic pattern analysis in research is one of the main uses for natural language processing, it is often used to analyze huge amounts of unstructured texts. Cai [3] has shown that the use of natural language processing (NLP) in urban studies has exponentially increased over the past decade. NLP is often used to analyze social media user content such as Tweets and Instagram comments, this was the case in 11 out of the 27 studies analyzed by Cai [3]. Besides entire sentence analysis these systems also use named entity recognition (NER) to analyze the unstructured text and quickly identify entities such as locational, temporal or organizational information from the text [4]. As can be seen in Helderop et al. [5] that used NER to analyze police reports and extract data from these reports to be used in further plotting and analysis.

NER and part of speech tagging both center around analyzing each individual word on its own. These individual word analysis tools are the main NLP functions that are used to understand the user queries provided to the navigational map-like system called Direct-Me, created by Withanage et al. [6]. Their system can direct users to a location or help them find a location based on user defined directions. Users can interact with the system using speech. The Direct me application pipeline first runs an automatic speech recognition algorithm on the recording, secondly uses natural language processing to linguistically analyze what the user said to then at last be able to understand the question and create a dependency graph to be used in the application to provide the user with directions to a location. The Direct-Me system is a highly specialized system which allowed its creators to make it really efficient for certain specific tasks.

Using the reviews that users leave on sites such as Yelp, the point of interest review question answering (POIReviewQA) system [7] uses these to train itself on the knowledge about these points of interests (POI's). Thus the system can answer questions about locations using the knowledge it gained from previous user to user question answering interactions and user

reviews. Places QA [8] expands on this idea by also incorporating the use of images found online of the point of interest. Places QA then analyzes these images using a convolutional neural network to see what is visible in them. It then uses a complex pooling algorithm to extract a conclusion and answer a question.

Yin et al. [10] have proposed a geospatial question answering system that uses NLP to understand questions and linguistically extract what data must be gathered. Using this data a visualization is created and returned to the user. Their system uses lookup tables to extract the entities needed to know what data must be gathered from the database. By using the lookup table results the data is then used to create a visualization which is returned to the end user with the specific points highlighted. Evaluation done by Yin et al. has shown that this system performs well and is intuitive for users to look at. Interestingly possible future research possibilities do highlight the potential use of artificial intelligence in the system.

2.1.2 Possible use of LLMs in Geospatial Sciences

Large language models (LLM's) are a specific type of model within natural language processing. They have been trained on a huge amount of natural language to be able to understand the world around us, the huge increase in training data and (transformers) architecture is what sets them apart from more traditional natural language processing systems [11]. There have been several projects that have used LLMs for geospatial sciences. Projects such as LLM-Geo, MapGPT and GeoLLM use the popular ChatGPT models developed by OpenAI.

SQL databases are often used to store geospatial data. Thus Jiang et al. [12] aimed at fine tuning an LLM to generate sql queries that can then be run against a database to get the necessary data. During fine tuning the LLM is given the SQL database structure schema and example data on question and query requests. From testing the clear limiting factor preventing chatgpt from becoming a geospatial analyst is the hallucination, as in complex queries requiring joins of multiple tables or datasets it still often incorrectly answers questions.

Punjani et al. [13] used a similar approach to create their GeoQA2 system. Interestingly their system was able to handle more complex queries as it generated sparQL queries that allowed for arithmetic operations to be embedded inside.

Given the large amount of understanding that LLM's have, the creators of LLM-Geo [14] attempted to set five goals that the LLM must be able to do on its own to achieve the status of autonomous AI-powered. On its own the LLM must be able to self-generate, self-organize, self-verify, self-execute and self-grow [14]. Their proposed architecture to achieve these five goals uses the proprietary gpt-4 model which has access to the entire internet, which allows it to find datasets online. These datasets are then analyzed and used to generate python code that creates the visualization for the end user.

Just like the LLM-Geo system the MapGPT [15] system found that GPT LLMs on their own are incapable of answering geospatial contextual questions on their own. Thus a training dataset containing both textual data and spatial information was proposed. Allowing the system to effectively integrate spatial data into its internal reasoning methods producing contextually aware responses when asked location based queries [15]. During post training inference a low level vector database which contains location and spatial text vectors which is directly referenced by the system in the tokenization phase to understand what data is needed to answer the question.

In contrast to the LLM-Geo and MapGPT systems which both aim to fine tune existing large language models for geospatial question understanding, the GeoGPT system [16] aims to create a system that knows how to use GIS tools such as buffer, intersect and erase. Using langchain a tool pool was created containing the tools and a short explanatory description of each. The GPT3.5 model from openAI was then able to reason and select what GIS tool to use. This tool could then lookup the data and the data would be used in the next inference step. This does rely on the transformer architecture being used.

2.2 State of the art

Traditionally natural language processing systems used user defined rules to analyze and understand text. Systems like NLP-QA created by Yin et al. [10] use lookup dictionaries to understand what data is needed to answer a question. More contemporary methods utilize statistical methods to interpret and understand the meaning of sentences and words. Named entity recognition is often used to identify spatial, locations and temporal terms in text as shown by Cai [3] and is also used in the “Direct Me” system implemented by Whithane et al. [6]. Current natural language processing systems are able to understand questions, they are unable to generate natural language answers back to the end user. Despite this limitation the well documented tools and fast algorithms that have been developed over the years do have a clear future in natural language processing and modules such as named entity recognition are still often used today as they are fast and easy to finetune. Current systems that use natural language processing such as direct me and POIReviewQA [6][7] are rudimentary however this is clearly reflected in their high accuracy as they tend to be very transparent and allow developers to tinker with every aspect of the system.

Large language models on the other hand are less transparent than NLP as the trained algorithms are like a black box where something goes in and something comes out, without exactly knowing what happens inside the system on a case by case basis [11]. Fine tuning however has provided a solution, allowing developers to provide the LLM with context about the situation that can be used to create an answer as seen in ChatGPT as a geospatial analyst [12] in which case the LLM was provided with database schemas and was prompted to generate sql queries. Other systems such as [16] have shown us that it is possible to receive fairly accurate answers when providing an LLM with the tools needed to answer questions. In the case where an LLM is given access to tools it can also reason accurately which tools it needs to use [16]. A clear drawback of using LLM's with geospatial data is hallucinations tend to happen [11][12]. At times the system hallucinates data points or answers that are not based on any data, this leads to confident answers that actually are incorrect.

Currently most large language model systems such as MapGPT [15] are not end to end natural language systems. They do take natural language in but usually output just pure data from a database or a visual representation of the data. Even in GeoGPT [12] which is an LLM that has data lookup tools, it outputs a visual shape and is not made to primarily output natural text. This is as most underlying GIS tools used in GeoGPT are not aimed at producing natural language results but instead produce visualizations with basic captions.

2.3 Reflection on background research

From the literature it is found that using natural language in geospatial sciences is a new but emerging topic. There are a limited amount of studies available on the use of natural language processing and large language models to interact with geospatial data. The studies that are available do highlight the need for a system that allows for interaction with geospatial data using natural language [10][12]. Currently there is a tradeoff between complexity of questions that a system can answer and the accuracy of its answers. A system such as the geospatial-QA system [10] that uses lookup tables consequently produces accurate results but uses rudimentary user defined rules and cannot answer questions that it has not seen before in a sense. A more open system such as the advanced GeoGPT [16] that uses large language models for reasoning can answer any type of question, also ones that it hasn't seen before. But due to the inherent use of a LLM for reasoning the system is not able to answer complex questions accurately [12].

LLMs have shown to be superior to NLP in geospatial question answering systems as they allow for more autonomy and natural language answer generation. But before being able to use an LLM for tool use reasoning with complex questions, more research needs to be done into the reasoning capabilities of open source models, as all systems researched use a proprietary model from a such as ChatGPT. From current research it is clear that closed source models are better at tool use and reasoning than open source models [17]. As of writing there is also a lack of literature and documentation on fine tuning open source models on tool use.

Chapter 3 - Methods and Techniques

From the previous chapter it is clear that an end to end natural language question answering system for geospatial data does not yet exist to the extent that is needed for Gaea. So in this chapter the design process of creating this system will be explained.

In this project a chatbot system is designed for the SuPerWorld research group, the main approach used was the co design approach. The co-design process is detailed in section 3.2. Besides co-designing the Creative Technology design process also played a key role in development, this is specified in section 3.2.

3.1 Co-design

In this project a chatbot like system is built for the online Gaea tool. This system will allow users to interact with geospatial data using natural language questions. As the SuPerWorld team (the team behind Gaea) already were experts in the field of geospatial science, co-design was a favorable method to develop this project.

3.1.1 Engaging & Understanding

In this first step of co-designing it was important to learn from the experts at SuPerWorld, they already had valuable experiences and knowledge in the field of geospatial science. The team members had also given the project some thought and explained their thoughts and ideas. They explained how they envisioned the use of modern LLMs in this system.

After getting to know the team they were able to adapt to my kinetic learning style to help me explore the topic of geospatial data by performing many smaller tasks that allowed me to better understand the issue at hand.

Once the problem and topic was understood the challenge could be set. From here research started by looking into current solutions of using NLP with geospatial data, as well as researching the state of the art. The sub-research-question: "How is natural language processing currently being used to interact with geospatial data?" was answered by performing a literature review. The results of these steps can be found in chapter 2, background research.

3.1.2 Ideation & Validation.

After acquiring a clear understanding of the topic at hand and gaining more insights into geospatial data and NLP the designing of concepts could start. Constant weekly meetings with the team allowed for quick validation of new ideas.

Once ideas became more concrete and positive feedback was received, prototypes could be built. Besides the in and output of a prototype, also the individual modules and models were discussed at the meetings. Allowing for direct feedback and constructive criticism from industry experts. This feedback would then be taken into consideration while redesigning and optimizing the prototype.

Questions written for system testing were also reviewed by creators of the datasets. Allowing quick and easy insights as to what users may want to ask questions about.

3.2 CreaTe Design Process

In this project the Creative Technology design cycle that has been developed by Eggink and Mader [18] (figure 3.2.1) served alongside the co-design framework as a clear basis. The Creative Technology design process consists of four steps: ideation, specification, realization and evaluation.

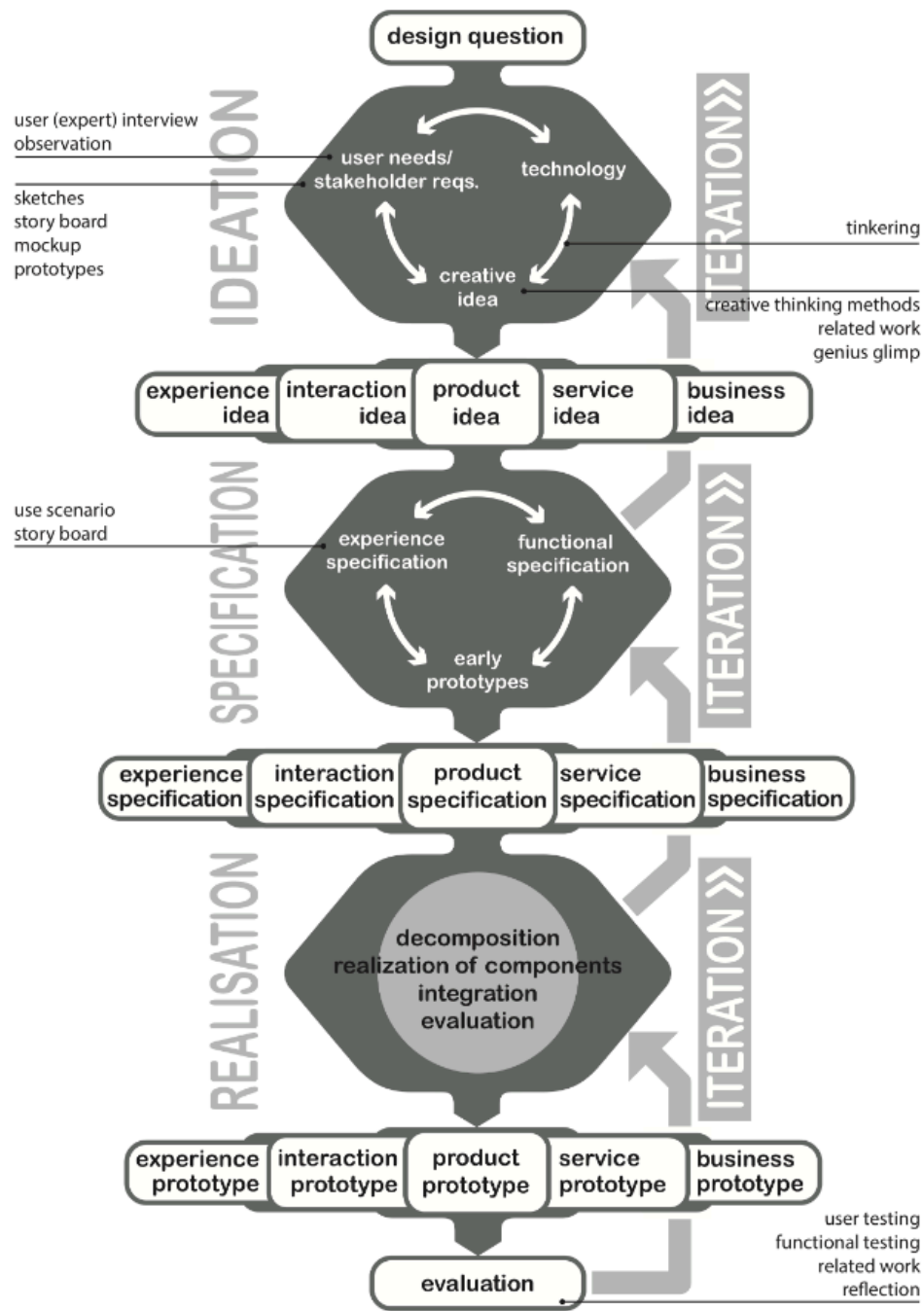


Figure 3.2.1: The creative technology design process. Source: [18].

3.2.1 Ideation

During the ideation phase the stakeholders and their needs were identified. Key stakeholders include the SuPerWorld team, end users and government agencies. Each stakeholder has their own priorities and it is important to understand these.

Ideation started with the exploration of geospatial sciences and tools commonly used for geospatial data analysis. Research also continued into natural language analysis and furthered into exploration on the current capabilities of NLP tools and LLMs.

Next, by creating sample questions the needs of the end users were evaluated, patterns were observed from these sample questions. These thematic patterns allowed for the processing of natural language questions from users. While creating these questions and assessing the needs of the stakeholders the first designs started being made. Co-designing allowed for easy and quick validation of designs that later morphed into early prototypes.

During ideation it became clear that not all questions are created equally. Some questions require a huge amount of data and complex comparisons. Simple questions on the other hand may only consist of a simple data lookup. Thus a ranking of seven increasingly more complex levels of questions was created to allow for specific complexity evaluation and clear scope definitions. More on the levels of complexity can be found in section 4.1.

3.2.2 Specification

During the ideation phase both NLP prototypes and LLMs that were given access to tools were used. These were both tested. The LLM system turned out to be poorly designed at first due to the use of the microservices architecture which resulted in inaccurate data and thus also inaccurate answers. So a step back was first made to the ideation phase to solve this issue.

Once a new system was built this was tested against the NLP system. A choice had to be made as to which system would be optimized. The NLP system was chosen as it allows for more tinkering during the optimization phase.

During the specification phase requirements were set. The key requirements were that the system must be fast and accurate. To allow for open access the system must not rely on 3rd party proprietary or paid technology, lastly the system must be easy to integrate.

3.2.3 Realization

During the realization phase of the Creative Technology design cycle the prototype system was built out for all the different services that Gaea offers. Each individual module was optimized for its purpose. The coherence between modules was also improved allowing for an increase in accuracy. The realized system was also deployed on a server, allowing for fast responses.

3.2.4 Evaluation

To quickly evaluate the system and iterate on new designs, an answer evaluation system was implemented that allowed users to rate their answers. The feedback received from these evaluations could be used to reiterate in the ideation or optimization phases using these new insights.

Lastly the project and system had to be evaluated. This was done by creating a list of questions and testing the system on these questions. The system has also shortly been in production. Allowing for data collection on real world usage. This data was also used to evaluate the answering speed which is one of the requirements. The other requirements were also assessed in this phase.

Chapter 4 - Ideation

4.1 Basic NLP System

The proposed NLP system uses the SuPerWorld GeoAPI to access necessary information to answer a question, thus it can only answer questions that the GeoAPI has data on [19]. For this reason it was chosen that a set of sample questions was to be created to give a deeper understanding into the different types of questions that users might ask. Initially a set of 43 questions were created relating to the weather domain (appendix A2). These questions were then manually annotated with their corresponding topic, category, if they require a temporal or spatial comparison, their location and date time if mentioned.

An example is: “What was the warmest month in Agros in 2020?”, this question is comparative temporally, the topic is “warmest month”, the location is Agros, the temporal range is 2020 and the category is “temperature”, because the /temperature API-endpoint would need to be queried to get the necessary data.

These 43 questions in appendix A2 were used to gain a firm understanding of the different levels of complexity and possible patterns in these questions. From these questions it was identified that the temporal and spatial values could be used as variables, thus these questions were reformed into a set of template questions (appendix A3), where the temporal and spatial value would be left as a variable which can be entered later. The question: “What was the warmest month in Agros in 2020?” would now become “What was the warmest month in [*location*] in [*date*]?”, where we can substitute in any location and date range to get a correctly formulated question.

To identify which API endpoint needs to be requested it must be known what the topic of the question is. Named entity recognition (NER) can be used for this, however an NER model needs to be trained on example data, in this case questions, the extracted topics and locations were used. To be able to do this the topic and location were annotated from each of the questions. For training an NER model a list of 150 location names in Cyprus was obtained and a list of a few different temporal ranges was created. These were then placed into the placeholders “[*location*]” and “[*date*]” respectively. Using this method we were able to extrapolate these 22 template questions into thousands of questions with their topic and location automatically annotated. These could then be used to train an NER model on recognizing the topic and locational information from a question. This model was able to accurately identify the location names in Cyprus from questions, even names that were not in the training data. It was also able to confidently extract topics from questions by using linguistic analysis.

Next when investigating and creating a template question set relating to the vicinity domain (appendix A4) it became clear that questions started to become repetitive, as the question “How far is [*location*] from the nearest road?” is linguistically alike “How far is [*location*] from the nearest beach?”. Thus another variable was introduced, the object of the sentence. This could be: “the beach”, “a natura 2000 location”, “the electrical grid” and “a road”. Questions were then

created that had an object placeholder inside of them, such as: “What is the distance from [*location*] to [*object*]?”. Because this domain does not support temporal variables these were left out.

Just as before these 10 questions were combined with the list of 150 location names and 5 different objects to create a list of 7500 questions. The inserted locations and objects were also automatically annotated, this dataset could then be used to train a new NER model on vicinity questions.

4.2 Endpoint Recognition

The basic NER system works, it recognizes the intent of a question and detects the word that describes the topic. This topic is then compared to a predefined dictionary to find what endpoint must be queried. From the question: “How warm is it in Limassol in August?” the topic “warm” is extracted. The system will have a predefined dictionary on topic words that describe the temperature such as: “temperature”, “hot”, “warm”, “cold” and “cool”. The topic “warm” is in this list so the system will automatically make a request to the /temperature endpoint.

This system requires a predefined dictionary of keywords pertaining to each endpoint. This also prevents the system from accepting open questions that it has never seen before. To solve this a model that could detect the endpoint needed to be developed.

The training data for this model would consist of a huge list of questions that now would have an endpoint associated with them instead of just an annotated topic. This list was then created containing [808 pairs of questions and their corresponding endpoint. More on the endpoint recognition system can be found in section 6.1.

4.3 Answer Generation

Once the endpoint and location are extracted a request can be made to the SuPerWorld GeoAPI to get the necessary data to answer the question. We can then pass the original question and data into a large language model and prompt it to answer the question using the provided data. After which it will output an answer based on the data. More on the use of the LLM can be found in section 6.5.

4.4 Complexities

The example and template questions in appendix A were also used to identify different difficulties in question answering. These questions can be placed in seven different levels of complexities. These are based on the amount of requests that need to be made to the Gaea GeoAPI and the difficulty of comparing the response data to formulate a natural language answer.

Firstly the simplest question type is a factual question where only a single request has to be made and little to no processing needs to be done, for example “What is the temperature data in

location?”. The system would simply return the temperature data for this location. A question that would have a temporal parameter would be more difficult as it requires the system to understand the data output from the GeoAPI and filter it to only give the output data relating to the time frame asked. An example of this is “What is the temperature in *location* during May?”.

Secondly, a single request to the GeoAPI returns data for a single geographical point, at some topics it may also return historical data. Thus questions that require comparison between the data from a single request are harder to answer. These are categorized as comparative single request questions. An example of this is “When was the hottest month in *location*?”.

When performing spatial comparisons multiple API requests need to be made to gather the information of multiple locations. Thus comparative dual request questions such as “During May 2020 was *location1* colder or *location2*?” are harder to answer for the system. These questions however can also cover a wider geospatial area such as a region. In this case it would become a comparative multi request question that would require three or more requests. The dual locational questions belong in the third category, wider area questions belong in the fourth category.

Lastly, the most complex question would be a multi disciplinary question that requires multiple requests to be made to different API endpoints. The question: “What is the largest risk for *location*?”, would require an individual request to get the risk score for each of the different risks that the GeoAPI supports, this type of question would fall into the fifth category. When also considering possible temporal or spatial ranges, then questions can get more complex when also requiring comparison between different locations or inside of a polygon area. The question “At which location on the island is there the highest risk of a geohazard?” Would require an analysis of the entire island for each of the possible geohazards. Thus these types of questions belong to the sixth level. The most complex questions would also have a temporal variable, these belong to the seventh category.

These taxonomies assume that it is increasingly difficult to generate accurate natural language answers when the comparative difficulty increases.

4.5 Autonomous API Calling

Ideally the system could reason and make its own choices on which API endpoint to call to get the necessary information to answer a question. Thus a tool use solution was explored. How can we give an LLM access to the API?

Using the Langchain agents module it is possible to create tools, explain them to the LLM and have it use them autonomously. Tools would be created for each endpoint, another tool would allow the model to convert any location into coordinates. These coordinates could be used to make an API request using the API endpoint tools. The json data would then be returned as an observation and the system would use that to further reason and answer the question.

Chapter 5 - Specification

5.1 Functional Requirements

The natural language processing question answering system for geospatial data must above all be end to end natural language. This entails that a natural language is input and also a natural language answer is output. This answer of course must be based on the reference API data.

Because a system like this may request incorrect data or make other mistakes, there must be a clear disclaimer warning visible to end users interacting with the system. It is important to remind users that this is an experimental tool and only meant for orientational purposes, users must always double check important information. The scope of this project is only to create an API and a sandbox web environment for testing thus this is not a hard requirement for this project now.

5.2 Non-functional requirements

The system must be accurate and provide correct answers to the questions that users have as often as possible. The system must have an accuracy of at least 90% in production, we must also give users the possibility to rate answers and mark incorrect ones. Allowing us to learn from our mistakes.

The goal of this project is to help and serve users as much as possible and leave them with a satisfied feeling as this leads to more use in the future. Gnewuch and Adam [20] clearly show that users' satisfaction of using the product decreases when they have to wait longer. Thus an answering time frame of 10 seconds was chosen. On average the system must answer questions within 10 seconds when running in production.

An important value that the SuPerWorld team, the team behind Gaea, believes in is that information must be open and available to all. Thus this project should not be built on paid proprietary software. The system should be completely self hosted and be run locally without the use of any external paid API's or services.

Lastly this system is not meant to be directly used by users, instead this API is meant to be integrated into the frontend of GAEA. Thus the API must have clear documentation on how endpoints work and interact with each other. There must also be clear error messages whenever possible.

5.3 Autonomous LLM vs. NLP system

As there are two different systems each with their own architecture a test was proposed to explore both systems deeper and choose which system better matched the requirements of the project.

For this test both systems, the natural language processing system and the autonomous tool calling a large language model were built. Each would support questions about five endpoints: nearest blue flag beaches, nearest roads, wildfire risk, temperature and wind data.

Once both systems were built out a set of 9 questions was created, three for each complexity level. Each system would be scored on if the correct locations were extracted and correctly converted into coordinates, if the correct endpoint was recognized and a request was correctly made and lastly if the natural language answer that was output was correct and based on the data.

From preliminary tests it quickly became obvious that the autonomous tool LLM struggled with understanding that it must first convert a location into coordinates and then use those coordinates, as in almost all cases coordinates used to make requests were hallucinated and were incorrect. This also led to the API providing data that was not for that location which then led to incorrect answers. In further testing and debugging it became clear that the system struggled with correctly analyzing the multiple steps that it would need to take and that multi step questions were exponentially harder for it to process.

To solve this issue a second version of the autonomous tool using LLM was created. The tools in the system would be integrated as much as possible to allow for as little tool use as possible. For each API endpoint tool the get coordinates function was now integrated into it. Instead of first needing to use a tool to convert the location into coordinates and then use those coordinates in another tool, now a more integrated tool was created allowing for direct input of a location of which the coordinates would automatically be used to get the data. Thus eliminating the possibility of hallucination by the LLM.

5.4 System Selection

Testing these three systems showed that the second optimized version of the autonomous tool LLM performed much better than the first one as can be seen in figure 5.4.1.

System Accuracy

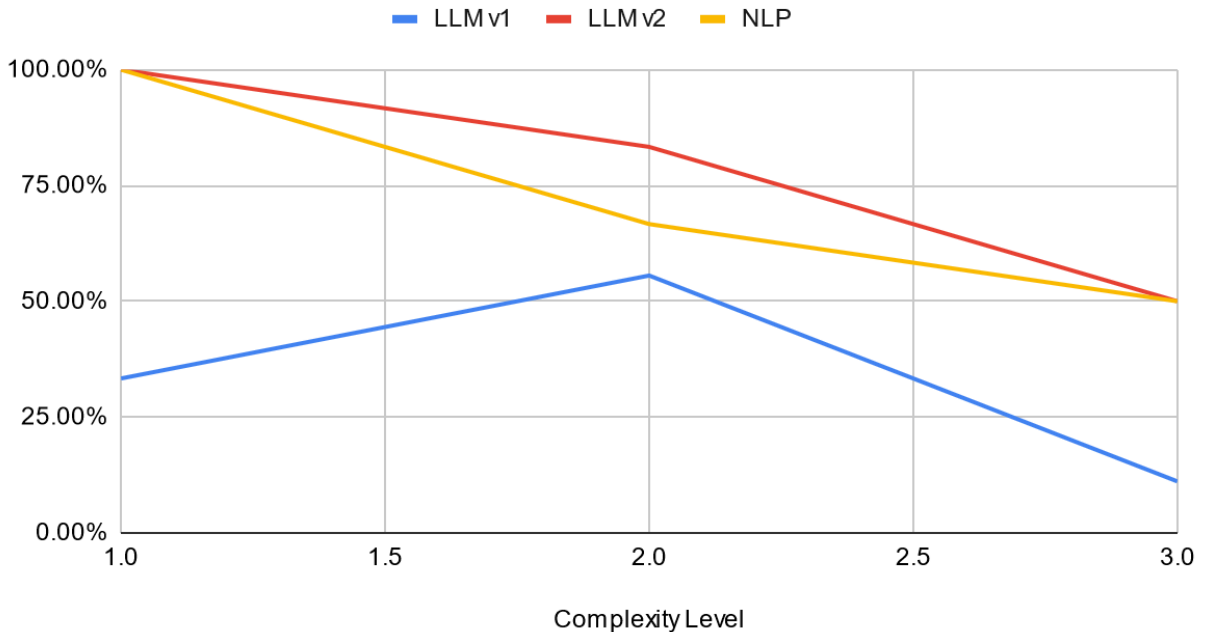


Figure 5.4.1: Results of comparing autonomous LLM V1 (blue), autonomous tool LLM V2 (red) and the NLP system (yellow) to each other.

Lastly a choice had to be made as to which system would be used. For this the NLP system was chosen, even though the system has a slightly lower accuracy in testing done in this experiment, the system does allow for more tinkering and fine tuning of individual modules. It also allows for greater expansion which the LLM system does not allow for. Tool calling with the open source Llama3 model is a new feature, thus the limited resources on it are focussed on inference use and not on specific fine tuning on certain tools. The NLP system is also faster on average compared to the LLM system that has to constantly iterate over itself every time it makes an API request.

Chapter 6 - Realization

After testing and comparing both the NLP system and the Toolled LLM system, the NLP system was chosen and built to production standards. The system consists of four individual modules that process information and help answer the question.

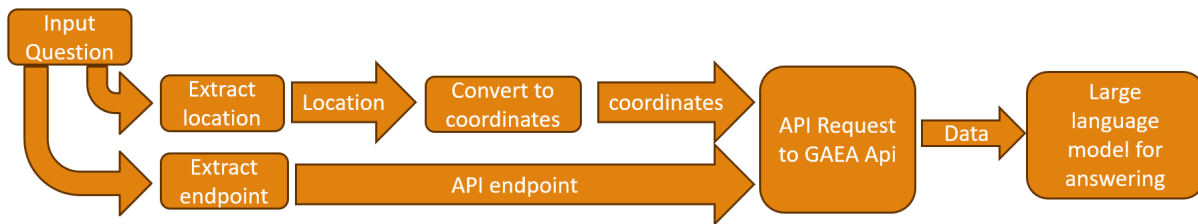


Figure 6.1: NLP system pipeline overview from question to final answer, showing the five intermediary steps.

The pipeline has been initialized in Python where flask is used to create a webserver. All code can be found on github [21].

The relationship between the individual modules visualized in figure 6.1 is programmatically shown in figure 6.2. The system starts with an input question which is then analyzed to extract the locational information and determine what endpoint is relevant. The locations are then converted into a pair of coordinates, after which requests are made to the Gaea GeoAPI. This data and the original question are then passed into an LLM tasked with generating a natural language answer.

```
def run(question):

    # get the endpoint
    endpoint = predict_endpoint(question)
    print(f"Predicted: {endpoint}")

    # Get locations
    locations = get_locations(question)
    print("Locations and addresses:", locations)

    # convert location to coordinates
    coordinates = []
    for location in locations:
        c = get_coordinates(location)
        coordinates.append(c)
        print(c , '<- ', location)
```

```

# make the request to the api
data = []
for coordinate in coordinates:
    d = request_data(
        endpoint = endpoint,
        coordinates = coordinate
    )
    index = coordinates.index(coordinate)
    print(index)
    data.append(f'Data on {endpoint} for {locations[index]}: {d}')

print(data)
answer = process_answer_data(question, data)
return answer

run('Is Paphos or Lefke closer to a beach?')

```

Figure 6.2: Global overview of how data is passed between the different modules and how modules interact with each other.

6.1 Endpoint prediction

Firstly an incoming question is analyzed to find out what data needs to be requested from the GAEA API to answer the question accurately. The `predict_endpoint` function is executed to use the Bert model [23] that has been trained on classification of questions into endpoints. More on the training of the model can be found in section 6.5.

```
def predict_endpoint(question):
    # Tokenize the input question
    inputs = endpoint_tokenizer(question, return_tensors="pt")

    # Pass the question as tokens forwardly to the model.
    outputs = endpoint_model(**inputs)

    # get the predicted class enumerator
    predicted_class = torch.argmax(outputs.logits).item()

    # map the enumerator to an actual endpoint from the dictionary
    predicted_intent_mapping = {v: k for k, v in label_mapping.items()}
    endpoint = predicted_intent_mapping[predicted_class]
    return endpoint
```

Figure 6.1.1: inference code for using the fine tuned Bert classification model to predict the endpoint relevant to that question. (Boilerplate code that was not deemed important for explanation has been left out; it can all be found in the Meliferea git repository [21]).

6.2 Location extraction

The locations of a question are extracted using the earlier trained NER model that previously served to also extract the topic of a question.

```
import spacy
nlp_ner = spacy.load('model-best')

doc = nlp_ner('How far is kalo chorio from the electrical grid?')
print(doc)

colors = {"location": "#FFCBCB", 'topic': "FD8A8A"}
options = {"colors": colors}

spacy.displacy.render(doc, style="ent", options=options, jupyter=True)
```

Figure 6.2.1: Using the 'model-best' that was trained the locations in questions are identified and visually shown using displacy (figure 6.2.2).

How far is **kalo chorio** **LOCATION** from the **electrical grid** **TOPIC** ?

Figure 6.2.2: Shows the output of the code in figure 6.2.1.

After converting the question string into individual tokens and the corresponding entity identifications, simple logic can be used to extract a list of locations from the question as can be seen in the code in figure 6.2.3.

```
# Get the location name
def get_locations(doc):
    locations = []
    for ent in doc.ents:
        if ent.label_ == 'LOCATION':
            locations.append(ent.text)
    return locations
```

Figure 6.2.3: Logic used to extract a list of locations from the question.

6.3 Conversion of location to coordinates

After we successfully identify the locations in a question we need to convert these locations into coordinates as the GeoAPI only accepts coordinates. In figure 6.3.1 the system loops through the locations list and converts each location into a set of coordinates using the `get_coordinates` function. It adds these to a new list 'coordinates'.

```
coordinates = []
for location in locations:
    c = get_coordinates(location)
    coordinates.append(c)
    print(c , '<- ', location)
```

Figure 6.3.1: Code that converts the locations into coordinates which is an excerpt out of the main `run()` code from figure 6.2.

The `get_coordinates` function (figure 6.3.2) then uses an external geocoding API provided by `locationIQ` [22] to convert the location into coordinates. It then returns a tuple containing the latitude and longitude, if it can find any in Cyprus for that location query.

```
def get_coordinates(query):
    # Encode the address
    encoded_address = urllib.parse.quote(query)
    api_access_token = "[**redacted**]"
```

```

# insert the API key and encoded location name into the url
url =
f"https://us1.locationiq.com/v1/search?key={api_access_token}&q={encoded_addresses}&format=json&"

# make the request
response = requests.get(url)
data = response.json()
# extract the coordinates if found
if (response.status_code == 200):
    return (data[0]['lat'],data[0]['lon'])
elif (response.status_code == 404):
    return f"Location: {query} does not exist!"

```

Figure 6.3.2: Using the locationIQ geocoding api the location name is converted into a tuple containing the latitude and longitude.

6.4 Requesting data from the GAEA API

Finally when we have the coordinates for all the locations in the question and know the endpoint, a request can be made to the Gaea GeoAPI. Just as when we get the coordinates, here we also loop through the locations list and make a request for each location as can be seen in figure 6.4.1

```

data = []
for coordinate in coordinates:
    # actually get the data from the api
    d = request_data(
        endpoint = endpoint,
        coordinates = coordinate
    )

    # calculate coordinate index
    index = coordinates.index(coordinate)

    # add results to the list
    data.append(f'Data on {endpoint} for {locations[index]}: {d}')

```

Figure 6.4.1: Excerpt from figure 6.2 that loops through all the coordinates and makes a request for each coordinate to get the relevant data then adds that data to the list.

The data list in figure 6.4.1 is directly passed into the LLM for natural language answer generation. Inside figure 6.4.1 the `request_data` function (in figure 6.4.2) takes care of authentication and requesting data from the Gaea GeoAPI.

```

import requests

def request_data(endpoint, coordinates):
    if endpoint == 'temperature':
        endpoint = endpoint.capitalize()

    # construct the url
    url =
f"http://213.7.195.74:8080/v2.0/{endpoint}?lon={coordinates[1]}&lat={coordinate
s[0]}"
    print(url)
    payload = {}
    headers = {
        'x-api-key': '[**REDACTED**]',
        'Authorization': '[**REDACTED**]'
    }
    # make the request
    response = requests.request("GET", url, headers=headers, data=payload)

    # check the response
    if response.status_code == 200:
        r = response.json()
        print(r)

        return r['data']
    else:
        return 'could not find any data'

```

Figure 6.4.2: Request data function that handles API requests to the GeoAPI and returns the json data.

6.5 Generating a natural language answer using an LLM

Lastly once we have all the data we can use an LLM to generate a natural language answer (figure 6.5.1). For this we use the Llama3 model running in Ollama as it is open source, fast and a highly capable model. Using Ollama also allows us to easily substitute in other models in the future.

```

answer = process_answer_data(question, data)
return answer

```

Figure 6.5.1: Excerpt from figure 6.2 that passes the data from the Gaea GeoAPI and the original question into the process_answer_data function.

Figure 6.5.1 shows the relationship between the output of the earlier stages and how the data is passed into the `process_answer_data` function. Figure 6.5.2 shows the inner workings of that function and direct use of the LLM through Ollama.

```
Llm = ChatOllama(model="llama3")

def process_answer_data(question,data):
    response = llm.invoke(f'You are a question and answering chatbot that
users use to ask questions about geospatial data. You will be provided with the
question and the data that came from a rest api. Please answer the question
given short and consisely based on the data! Do not provide any internal
context or any other thoughts that cannot be verified using the data from the
request! Please answer the question {question} using the information: {data}')
    return response.content
```

Figure 6.5.2: Inside the `process_and_answer` function the LLM is started and then for every call it is prompted to use the data and original question to generate a natural language answer.

In figure 6.5.2 the question and the data are inserted into a premade prompt that gives the LLM context as to what its task is. The prompt also tries to prevent hallucination by instructing the LLM to answer using the data.

6.6 Training the endpoint prediction module

From a question the endpoint is predicted using a fine tuned Bert classification model that is based on the bert-base-uncased pretrained model [23]. The corresponding tokenizer from the pretrained bert-base-uncased is also used.

This section will highlight important code and explain it for a global understanding of how this model was created. For reproducing, the full code and a jupyter notebook can be found in the github repository [21].

```
datasets = [
    'datasets/endpoint/weather.txt',
    'datasets/endpoint/vicinity.txt',
    'datasets/endpoint/risk.txt',
    'datasets/endpoint/geohazards_other.txt',
    'datasets/endpoint/geo_attributes.txt'
]

questions = []
endpoints = []
```

```
for dataset in datasets:
    csv_dataset_path = dataset
    dataset = pd.read_csv(csv_dataset_path)

    # Split the dataset into a list of questions and intents
    questions.extend(dataset['question'].tolist())
    endpoints.extend(dataset['endpoint'].tolist())

print(len(questions))
```

Figure 6.6.1 : Datasets for each category of endpoints are imported (due to the length of the dataset these have been left out of this report, but can be found at the github repository [21]). These datasets are merged together and split into two lists, the first containing the questions and the second containing the endpoints.

```

label_mapping = {
    'temperature': 0,
    'precipitation': 1,
    'wind': 2,
    'humidity': 3,

    'distance-sea': 4,
    'nearest-blueflag-beach': 5,
    'natura-region': 6,
    'distance-electric-grid': 7,
    'distance-road': 8,

    'subsidence-risk': 9,
    'landslides-risk': 10,
    'wildfire-risk': 11,
    'seismic-risk': 12,
    'flooding-risk': 13,

    'seismic-Zone': 14,
    'geo-Suitability-Zone': 15,
    'burned-area': 16,

    'aspect-slope': 17,
    'elevation': 18,
    'near-tree': 19,
    'vegetation': 20
} # map enumerators to labels

labels = []
for endpoint in endpoints:
    if endpoint in label_mapping:
        labels.append(label_mapping[endpoint])
    else:
        raise ValueError(f"No mapping found for endpoint: {endpoint}")

```

Figure 6.6.2: To simply classification the different possible endpoints are mapped to an enumerator. Each endpoint in the list is then replaced with its corresponding number.

```

train_inputs, val_inputs, train_labels, val_labels =
    train_test_split(
        encoded_inputs['input_ids'],
        labels,
        test_size=0.2,
        random_state=42)

```

Figure 6.6.3: The dataset is then split into a training dataset and a validation dataset. 80% of the questions are for training and 20% are for validation. A fixed random state was used to allow for easy reproduction of the results.

```

# Training loop
num_epochs = 100

for epoch in range(num_epochs):
    model.train()

    # loop through the dataloader batches
    for batch in train_dataloader:
        # train and go forward
        input_ids, labels = batch
        optimizer.zero_grad()
        outputs = model(input_ids, labels=labels)
        loss = outputs.loss
        loss.backward()
        optimizer.step()

    # Validation loop
    model.eval()
    val_loss = 0.0
    correct_predictions = 0

    with torch.no_grad():
        # loop through batches in the validation dataloader
        for batch in val_dataloader:
            # run validation inference, check and count correct
            input_ids, labels = batch
            outputs = model(input_ids, labels=labels)
            val_loss += outputs.loss.item()
            logits = outputs.logits
            predictions = torch.argmax(logits, dim=1)
            correct_predictions += torch.sum(predictions == labels).item()

# Calculate validation metrics

```

```

val_loss /= len(val_dataloader)
accuracy = correct_predictions / len(val_dataset)
print(f'Epoch {epoch + 1}/{num_epochs}, Val Loss: {val_loss:.4f}, Accuracy:
{accuracy:.4f}')

```

Figure 6.6.4: Training and evaluation loop. (hidden: conversion of the dataset into the dataloader done before training)

```

# Save the fine-tuned model
save_path = f'models/endpoint/model-e{num_epochs}-{accuracy:.2f}'
model.save_pretrained(save_path)
tokenizer.save_pretrained(save_path)

```

Figure 6.6.5: Lastly the model needs to be saved, this is done in the models/endpoint folder. The model name is dependent on the amount of epochs and the last accuracy. This is for easy identification later.

Figure 6.6.1 up to 6.6.5 globally show how the model was trained. Examples on inference can be seen in section 6.1. All code and example notebooks can be found on github [21].

6.7 Training location recognition model.

To know for what location a request needs to be made, the system identifies the locations in questions using a custom made NER. The NER model was first also used to identify topics of questions, however this part has been replaced by the endpoint classification bert model as shown in section 6.6.

First the template questions list ('template_questions.csv') and the list of locations (town_and_cities.csv) are imported. The different locations are then substituted into the template questions. This is shown in figure 6.7.1. (Note: the code has been condensed for readability.)

```

# import CSV's and convert to lists
csv_questions = open('template_questions.csv')
csv_tq = csv.DictReader(csv_questions)

tac = open('towns_and_cities.csv')
csv_tac = csv.DictReader(tac)

# insert topics and locations into questions
for q in csv_tq_list:
    for t in possible_topics:
        question = q.replace('[*object*]', t)

```

```

for l in csv_tac_list:
    new_question = question.replace('[*location*]', l)

# append to lists
    questions.append(new_question)
    locations.append(l)
    topics.append(t)

```

Figure 6.7.1: Code that imports the csv's that contain the questions and items that are substituted in. Finally questions and annotations are added to their respective lists. (hidden: conversion from dictionary to lists and definition of lists).

Next the data is preprocessed, all letters are decapitalized and words such as 'a' or 'the' are removed from annotations to prevent overfitting on the recognition of those words. All words are de-capitalized as users might not always take the time to capitalize pronouns, thus the model must not rely solely on capitalization of pronouns to detect location names.

Next the data in the lists is converted into the annotation format that is required for spacy. A dictionary list is made that contains the question, location, starting and ending character number of the location in the sentence, topic, the starting and ending character of the topic in the question. This is done programmatically in figure 6.7.2 for each question.

```

training_data = []
for question, topic, location in zip(questions, topics, locations):
    #print(question, topic, location)

    # Get location for topic
    t_start = question.find(topic)
    t_end = t_start + len(topic)

    # Get location for the location
    l_start = question.find(location)
    l_end = l_start + len(location)

    training_data.append({
        "question": question,
        "location" : location,
        "l_start" : l_start,
        "l_end": l_end,
        "topic" : topic,
        "t_start" : t_start,

```

```
"t_end" : t_end
})
```

Figure 6.7.2: Converting the data in the lists to a format that spacy can use to fit the model.

This data is then used to create an annotated doc for each example. These are then added to a Spacy doc bin which is saved. In figure 6.7.3 this has been done programmatically.

```
for entry in training_data:
    # Create a spacy doc for the question
    doc = nlp.make_doc(question)

    # list for the annotated entity labels
    # spans will be placed in here
    ents = []

    # Create a span object for the LOCATION annotation
    span = doc.char_span(entry['l_start'], entry['l_end'],
label="LOCATION", alignment_mode='contract')
    if span is not None:
        ents.append(span)

    # Create a span object for the SUBJECT annotation
    span = doc.char_span(entry['t_start'], entry['t_end'], label="TOPIC",
alignment_mode='contract')
    if span is not None:
        ents.append(span)

    if ents:
        filtered_ents = filter_spans(ents)

        doc.ents = filtered_ents
        doc_bin.add(doc)

doc_bin.to_disk('train.spacy')
```

Figure 6.7.3: Converting each entry into an annotated Spacy doc. These are added to a docbin which is saved for training.

Before training can start a config file needs to be created. For this the online tool at spacy.io was used [28].

Using the config file and the 'train.spacy' file training of the NER model can start. It is saved to model-last and if it is the best until now it is also saved to model-best.

```
!python -m spacy train config.cfg --output ./ --paths.train ./train.spacy --paths.dev ./train.spacy

i Saving to output directory: .
i Using CPU

===== Initializing pipeline =====
✓ Initialized pipeline

===== Training pipeline =====
i Pipeline: ['tok2vec', 'ner']
i Initial learn rate: 0.001
E   #      LOSS TOK2VEC  LOSS NER  ENTS_F  ENTS_P  ENTS_R  SCORE
-----
0     0          0.00    59.20    0.00    0.00    0.00    0.00
0    200         5.41   775.66  100.00  100.00  100.00    1.00
0    400         0.00     0.00   100.00  100.00  100.00    1.00
1    600         0.00     0.00   100.00  100.00  100.00    1.00
1    800         0.00     0.00   100.00  100.00  100.00    1.00
2   1000        0.00     0.00   100.00  100.00  100.00    1.00
3   1200        0.00     0.00   100.00  100.00  100.00    1.00
3   1400        0.00     0.00   100.00  100.00  100.00    1.00
5   1600        0.00     0.00   100.00  100.00  100.00    1.00
6   1800        0.00     0.00   100.00  100.00  100.00    1.00
✓ Saved pipeline to output directory
model-last
```

Figure 6.7.4: Using the training config file (config.cfg) and the training files to train the NER model.

For easy inference the displacy package is used to annotate and highlight the predicted location and topic of a question. The code for this can be seen in figure 6.7.5. In here the example question: "How far is kalo chorio from the electrical grid?" is run. In figure 6.7.6 the output can be seen, the location and topic have accurately been identified in this example.

```
import spacy
nlp_ner = spacy.load('model-best')

doc = nlp_ner('How far is kalo chorio from the electrical grid?')
print(doc)

colors = {"location": "#FFCBCB", 'topic': "FD8A8A"}
options = {"colors": colors}

spacy.displacy.render(doc, style="ent", options=options, jupyter=True)
```

Figure 6.7.5: Using displacy during inference for visualization. Output can be viewed in figure 6.7.6.

How far is **kalo chorio** **LOCATION** from the **electrical grid** **TOPIC** ?

Figure 6.7.6: Output of the code in figure 6.7.5, shows the location and topic that the system identified in the question.

In figure 6.7.1 until 6.7.6 a basic system can be seen, explaining how the NER system works. Please note this is not the end system that is used in production, that can be found in section 7.2.

Chapter 7 - Optimization

After building out the basic system as seen in chapter 6 it became clear that individual modules needed to be optimized, to perform better on their own but to also integrate and work together with other modules more effectively. Thus this chapter focuses on each module and the optimizations done to make the system production ready.

7.1 Endpoint extraction

Firstly the endpoint extraction module had to be expanded to serve almost all different services that GAEA has data on (figure 7.1.1). For this more datasets needed to be created. In total a list of 808 questions and the corresponding endpoints was made. This was used to train the classification model.

21 out of the 26 services that Gaea offers were integrated into this system. It was chosen that several land cover monitoring services would not be integrated as there was not a well defined API endpoint or the data was hard to understand for an LLM. Furthermore the endpoint providing the nearest amenities was also left out as the questions can be about a variety of different amenities as it serves many different points of interests. The amenities endpoint also returns a huge amount of data which for an LLM is too much to make sense out of. A custom data processing function could solve this in the future.

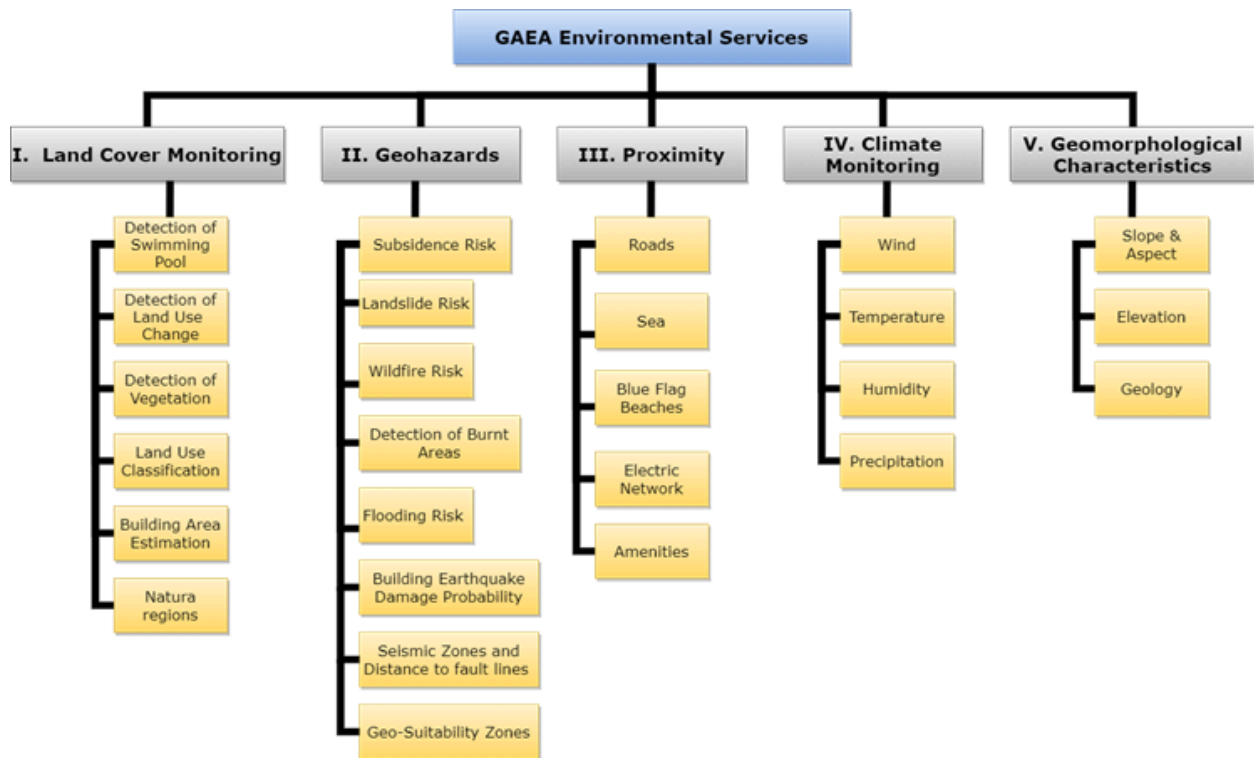


Figure 7.1.1: Different datasets that GAEA offers to users. From A. Jamil et al [1]

7.2 Extract location

In a lab setting the NER model that recognizes locations performs well during testing and accurately detects the location in a question. However when testing the model on real world locations and street names the model struggles to identify locational information such as house numbers, postal codes and (multi word) street names. The questions that the model has been trained on only consist of town names. A list of 150 town and city names were used for this. Thus the dataset did not accurately portray the real world situation.

At this point a new dataset could be created containing all the locations in Cyprus, however this would take considerable time to create, taking away resources from the greater scope of this project. Thus after researching, open source models were found that others had pre trained to recognize locational information from texts. The fastest and most reliable tokenizer that was also built to identify locational information was the Bert tokenizer. As the overarching Bert model is easy to fine tune, there are a plethora of different fine tuned models available from the open source community that each are specialized and fine tuned on their own tasks.

From preliminary testing three fine tuned models stood out:

1. Bert-base (NER) [24][25]
2. Bert-fine tuned [26]
3. Bert-large (NER) [27]

To determine the most capable model from the these three models, they were tested against each other. A set of 350 example questions that had the correct location annotated were used. These questions were split up into four categories based on the format of the location: single town or city names, street name + house number, street name + city name and street name + postal code. Each model was tested for questions in each of these categories.

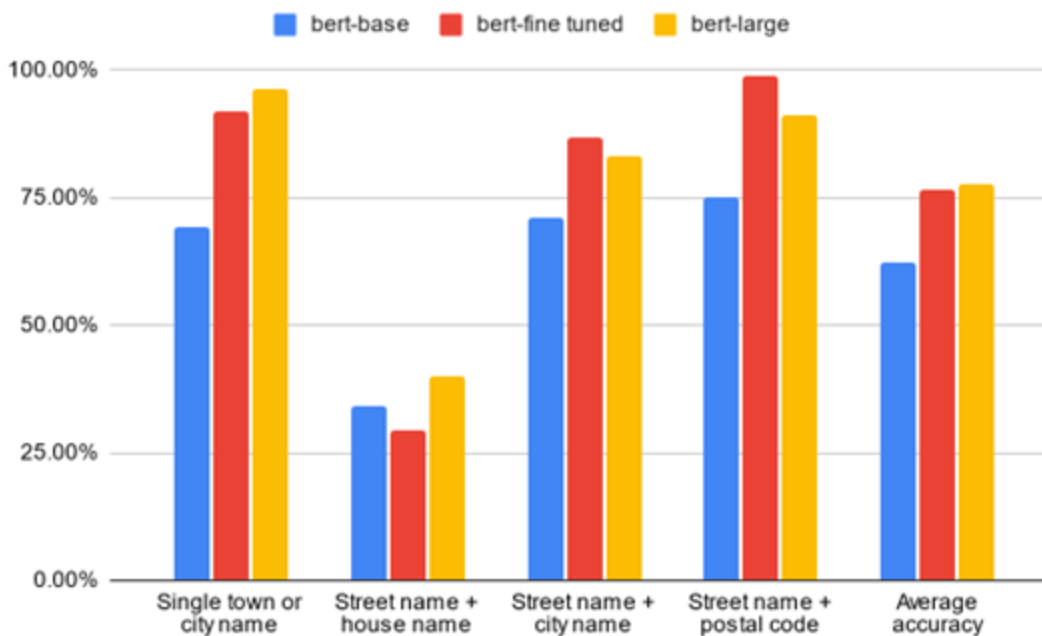


Figure 7.2.1: Results from testing three fine tuned models on four different locational formats. Average accuracy per model is also shown.

In figure 7.2.1 the results from these tests can be seen. Interestingly all models underperformed in the second category: street name + house number. After more investigation it became clear that the system struggles to recognize parts of street names such as “avenue”, “street” or “lane”. All NER models struggle with recognizing these words as part of the locational information.

Given the results from figure 7.2.1, the bert large model was selected to be used in the production system. Thus the module in the pipeline also had to be modified. A class was created to enclose the model (figure 7.2.2), this class also allows for easy inference.

```
class Locations:

    def __init__(self):
        # Load the model
        self.location_classifier = pipeline(
            "token-classification",
            "dbmdz/bert-large-cased-finetuned-conll03-english",
            grouped_entities=True,
        )

    def extract(self, question):
        d = self.location_classifier(question)
        locations = []
```

```

for l in d:
    locations.append(l['word'])

return locations

```

Figure 7.2.2: Location detection model class that (down)loads the model on initiation. The text can be passed into the extract function to extract a list of different locations.

This new system of extracting locations from questions now allows the pipeline to understand a much wider variety of formats that locations might be in.

Because Bert tokenizes texts not by individual words but by parts of words it can happen that not an entire word is recognized as a location. This part of word tokenization creates awkward situations where because only a part of the word is recognized as a location the system cannot find any coordinates associated with the half word.

Given the question: “What is the geo suitability of Kolokotroni Nicosia?”, the model should output: ‘Kolokotroni Nicosia’ as the only location. Instead the system outputs that it has detected two locations: ‘Kolokotroni Nico’ and ‘###sia’. Converting both of these strings into coordinates using the `get_coordinates` function will not work and result in no coordinates for both locations.

To solve this some clever logic has been added. Whenever the system cannot find anything for the individual location names it will first try to combine them all together and attempt to see if it can find coordinates for those locations. The logic of this has been implemented programmatically in Figure 7.2.3.

```

# If nothing is found then try combining
if all(item in ((0.00, 0.00), None) for item in coordinates):
    # Place all locations without hashtags in combine list
    combined_location = ''
    for location in locations:
        edited_location = location.replace('#', '')
        combined_location += edited_location
    # Add space before capital letters
    combined_location = re.sub(r'(?<!^)(?=[A-Z])', ' ', combined_location)

    # run coordinate check for combined list
    c = get_coordinates(combined_location)
    coordinates.append(c)
    locations.append(combined_location)

```

Figure 7.2.3: Programmatically combine all locational names together when nothing is found using the coordinate search.

However, at times when a question explicitly mentions two locations such as the question: “*Is Paphos or Limassol located at a higher elevation?*”, then combining the two locations will not result in correct coordinates. Thus if we still can't find any coordinates a third attempt will be made to split the combined string in half and check if this gives us any coordinates. Figure 7.2.4 shows the programmatically implementation of this logic.

```
# if still nothing is found then try splitting the combined string in half
if all(item in ((0.00, 0.00), None) for item in coordinates):
    # Check if combined location consists of two or more words
    combined_words = combined_location.split()
    if len(combined_words) >= 2:
        mid_index = len(combined_words) // 2
        first_half = ' '.join(combined_words[:mid_index])
        second_half = ' '.join(combined_words[mid_index:])

        # get coordinates, add to lists for first half
        c = get_coordinates(first_half)
        coordinates.append(c)
        locations.append(first_half)

        # do the same for the second half
        c = get_coordinates(second_half)
        coordinates.append(c)
        locations.append(second_half)
```

Figure 7.2.4: When we still can't find a location then try splitting the combined string from figure 7.2.3 in two and try again.

If none of these extra logic steps work we must inform the user that we have detected locations in their query, but we can't find any coordinates in Cyprus for these locations. We do this by returning a simple message and advising people to capitalize locational pronouns, shown in figure 7.2.5.

```
# If we still cant find anything, then return an error message to the user
if all(item in ((0.00, 0.00), None) for item in coordinates):
    ret_data = {
        'success': False,
        'answer': f'The locations {api_locations} were identified but we could
not find these in Cyprus. Please try capitalizing the names or providing more
information such as city or region.',
    }
```

Figure 7.2.5: When both logic approaches shown in figure 7.2.3 and figure 7.2.4 don't yield any coordinates for locations in Cyprus we notify the user and advise them to try capitalizing locational pronouns.

7.3 Converting the location to coordinates

As the GeoAPI from Gaea only accepts locational information as coordinates so we must first convert the location extracted earlier into a set of coordinates. A geocoding API is used for this. As there are many geocoding API's available a few were examined and the choice was made to use the LocationIQ API [22] as it has one of the best rate limits and in testing consistently produced accurate results.

The LocationIQ system is ratelimited to 2 requests per second. As we want to keep response times under ten seconds it is essential that this step should take as short as possible. Thus a caching system was set up inside the `get_coordinates` function.

When a call is made to the `get_coordinates` function with a location, that location is first checked against a database of known cached coordinate pairs. This simple database only has three fields: `timestamp`, `location_name` and `coordinates`. Whenever we can't find the coordinates in the internal database we make a request to the geocoding API. We then save these newly found coordinates in the database together with the location name. Thus in the future we can reuse this API request and not have to make it again, saving us precious time. Besides the location name and the resulting coordinates, the timestamp of the cached result is also saved, allowing us to retire and refresh the cache after a certain amount of time. The `get_coordinate` function with caching system can be seen well in figure 7.3.1.

```
def get_coordinates(query):
    # Check if coordinates are cached
    cached_coordinates = dbcl.get_cached_coordinates(query)
    if cached_coordinates != None:
        return cached_coordinates
    else:
        # if not then request the api
        encoded_address = urllib.parse.quote(query)
        api_access_token = "[**REDACTED**]"
        url =
f"https://us1.locationiq.com/v1/search?key={api_access_token}&q={encoded_addresses}&format=json&"
        response = requests.get(url)
        data = response.json()
        if (response.status_code == 200):
            found = False
            # loop through the results to find the top result in cyprus
            for loc in data:
```

```

    if (found == False) and ('Cyprus' in loc['display_name']):
        # add result to caching db
        dbcl.add_cached_coordinates(
            location=query,
            coordinates=(loc['lat'],loc['lon'])
        )
        # return coordinate set to program
        return (loc['lat'],loc['lon'])
    # if no coordinates can be found return none
    return None
elif (response.status_code == 404):
    # if no coordinates can be found return none
    return None

```

Figure 7.3.1: Programmatically convert a location name into coordinates by first checking the caching database, else making a request to the external geocoding API.

7.4 Requesting data from the Gaea GeoApi

After the coordinates and the endpoint are known a request can be made to the Gaea GeoAPI. The API provides us with the data needed to answer the question. The API that the production system relies on for its data can go offline. In this case the LLM will receive no response data. Results from testing with the GeoAPI offline show that about half the time the LLM will return a clear message stating that no data is found, other times it will return a hallucinated answer that is completely incorrect and based on nothing.

To prevent this hallucination an `api_offline` mode is made. So that when the API is offline the system will always return a message stating that part of the backend is offline and questions cannot be answered at this moment.

```

if gaea_api_down == True:
    ret_data = {
        'success': False,
        'answer': "I can't answer your questions now because the api
that I use to get information is down. Sorry for the inconvenience, please
try again later..."
    }
    return ret_data

```

Figure 7.4.1: If the api is down and unusable, the system always returns an error stating that it cannot get any information. This is done to prevent hallucination by the LLM due to missing data.

To pass the API response data into the LLM for answering the data must be explained, as some questions need multiple requests we need to clearly distinguish which API response belongs to

which request. Thus the data is placed into a preformatted sentence: “Data on {insert endpoint} for {location name}: {api response data}”. This sentence is then added into a list containing all the response data as can be seen in figure 7.4.2. This list is then passed into the LLM for answer generation.

```
data.append(f'Data on {api_endpoint} for {api_locations[index]}:
{requested_data}')
```

Figure 7.4.2: Explaining what the data from the API is on by mentioning the endpoint that it is from and the location for which it is, after which it is added to the data list.

From testing it turns out that the LLM struggles with generating answers for questions if there is a lot of data available. An LLM also struggles with making accurate comparisons between data points. If we were to take the example question of “Is Lefke or Agros closer to a beach?”. The classification system would identify that this question is about a beach and thus make a request to the “/nearest-blueflag-beach” endpoint. Instead of showing the nearest beach it returns the top three nearest beaches. This can be confusing for the LLM as it cannot accurately understand the difference between the three in the results.

To solve this problem custom data filtering rules were implemented. These user defined rules filter the response data and limit the amount of response data that is returned. In figure 7.4.3 a custom data filtering rule has been implemented to only return the first item when the endpoint on the nearest beaches has been queried.

```
# custom filter per endpoint
if endpoint == 'nearest-blueflag-beach':
    return r['data'][:1]

.....

else:
    return r['data']
```

Figure 7.4.3: User defined custom data filtering rules implemented to prevent and overload on data for the answering LLM.

7.5 Generating a natural language answer

Lastly when the data is known a natural language answer can be generated for the end user. This is done by passing the data and the original question into the Llama 3 model and prompting it to answer the question using the data.

To optimize the use of the LLM, four different prompt engineering techniques were tested to find out in what format the original question and data could best be passed to the LLM for answer generation.

All examples can be found in the git repository in: *"notebooks/llm-prompt-engineering.ipynb"* [21].

7.5.1 Prompt Structure 1

The first structure that was tested is a simple prompt just containing the question and response data from the API . This prompt also had a brief contextual explanation (Figure 7.5.1). The results from this approach are shown in figure 7.5.5 as structure 1 (blue).

```
def process_answer_data(question,data):
    response = llm.invoke(f'You are a question and answering chatbot that
users use to ask questions about geospatial data. You will be provided with
the question and the data that came from a rest api. Please answer the
question given short and concisely based on the data! Do not provide any
internal context or any other thoughts that cannot be verified using the
data form the request! Please answer the question {question} using the
information: {data}')
    return response.content
```

Figure 7.5.1: A simple prompt explaining the context of geospatial answer generation with the user question and API response data plugged in.

7.5.2 Prompt Structure 2

As seen in section 7.4, in testing the LLM struggles to make sense of the API response data when there is a huge amount of text. Thus the contextual explanatory instructions were split from the original question in the second structure. The prompt was split into several messages that were passed in as conversational chat history. In figure 7.5.2 two messages have been created. The first being the system message that gives the LLM context as to what its function is, the second human message informs it about the question and the data that it can use for answering.

```
def process_answer_data(question, data):

    messages = [
        SystemMessage(content="You are a question and answering chatbot
that users use to ask questions about geospatial data. You will be provided
with the question and the data that came from a rest api. Please answer the
question given short and consisely based on the data! Do not provide any
internal context or any other thoughts that cannot be verified using the
data form the request!"),
```

```

        HumanMessage(
            content=f"Answer the question {question} Using the data {data}"
        )
    ]

    chat_model_response = llm.invoke(messages)

    return chat_model_response.content

```

Figure 7.5.2: Using conversational chat history to split the instructions from the user question and data that is to be used.

7.5.3 Prompt Structure 3

To make the original user question even more clear a system was tested where the question on its own is shown to the LLM as a user message, we then pretend that the LLM has made its own choice to find the data after which it uses it to answer the question. (figure 7.5.3)

```

# Process question and data from request into a normal answer
def process_answer_data(question, data):

    messages = [
        SystemMessage(content="You are a question and answering chatbot
that users use to ask questions about geospatial data. You will be provided
with the question and the data that came from a rest api. Please answer the
question given short and concisely based on the data! Do not provide any
internal context or any other thoughts that cannot be verified using the
data form the request!"),

        HumanMessage(
            content=f"{question}"
        ),
        SystemMessage(
            content=f"To answer the question I have found following data:
{data}. I will use this to answer the question.",
        ),
    ]

    chat_model_response = llm.invoke(messages)

    return chat_model_response.content

```

Figure 7.5.3: Use a conversational chat history to pass a contextual system message, user question and the data needed to answer the question.

7.5.4 Prompt Structure 4

Lastly, to reinforce that the LLM uses the data to answer the questions and prevent hallucination or incorrect data use, an extra human message is used. In figure 7.5.4 we pretend that the LLM finds the data on its own after which it asks to verify it. We tell it that it is correct and that it must use it to answer the question.

```
def process_answer_data(question, data):

    messages = [
        SystemMessage(content="You are a question and answering chatbot
that users use to ask questions about geospatial data. You will be provided
with the question and the data that came from a rest api. Please answer the
question given short and consisely based on the data! Do not provide any
internal context or any other thoughts that cannot be verified using the
data form the request!"),

        HumanMessage(
            content=f"{question}"
        ),
        SystemMessage(
            content=f"To answer the question I have found following data:
{data}, is it correct?",
        ),
        HumanMessage(
            content=f"Yess the data is correct. Please use it to answer the
question"
        ),
    ]

    chat_model_response = llm.invoke(messages)

    return chat_model_response.content
```

Figure 7.5.4: Using human and system messages to pass all the data into the LLM for answering. Using more human messages to reinforce the importance of using the API response data for answer generation.

7.5.5 Comparing prompt structure

The four different structures shown in figures 7.5.1 up to 7.5.4 were compared by passing each a series of pairs containing a question and the data to answer that question. The question and

data pairs can be found in appendix C. For each question the data was already gathered and each system got the exact same question and input data. The questions were also ordered based on their respective complexity level, these levels were determined in section 4.4.

Accuracy for different prompt formats

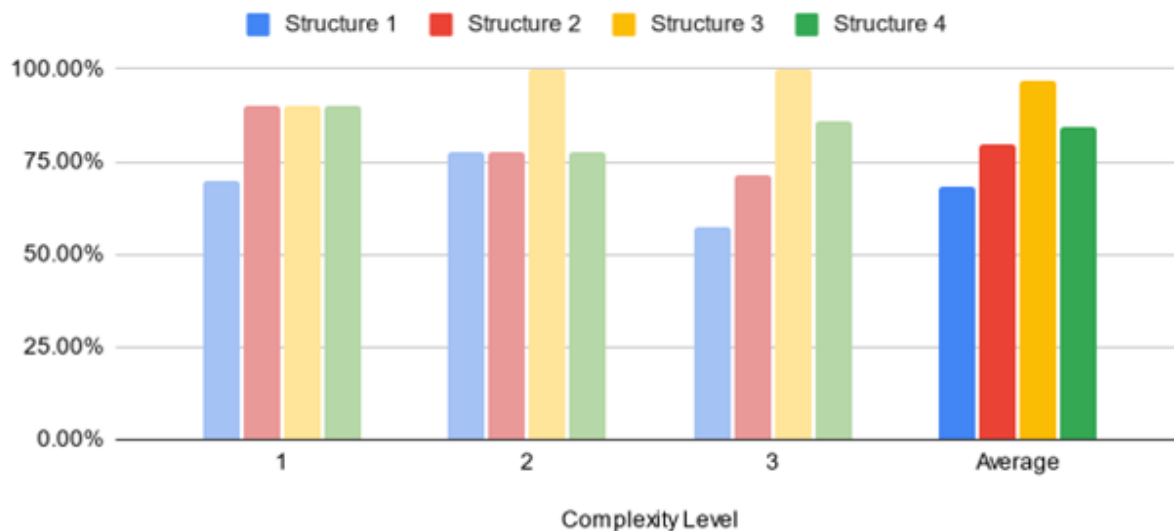


Figure 7.5.5: Results of testing the different answer generation approaches shown in figure 7.5.1 up to figure 7.5.4. The code in figure 7.5.1 is shown as structure 1 (blue), the code in figure 7.5.2 is shown as structure 2 (red), the code in figure 7.5.3 is shown as structure 3 (yellow) and lastly the code in figure 7.5.4 is shown as structure 4 (green).

The answer results in figure 7.5.5 clearly show that there is a benefit as to researching the different prompt engineering structures, as the previous structure used in preliminary testing (structure 1) has the lowest overall score. Structure 3 on the other hand shows us that by using conversational memory with human and system messages without explicit verification the system performs the best.

Following the results from 7.5.5, structure 3 was chosen to be used in production.

7.6 API Wrapper & Chatpage

To allow for easy and fast integration of the production system, the python `run` function from figure 6.2 (which encloses the entire pipeline from question to natural language answer) was made into an API endpoint using flask.

Using flask the application was converted into an API endpoint. Systems could now make requests to the `/query` endpoint, then after about ten seconds it would return an answer in json.

```
@app.route('/query')
def question():
```

```
request.start_time = datetime.utcnow()

args = request.args
question = urllib.parse.unquote(args['q'])

answer = run(question)

return answer
```

Figure 7.6.1: Using Flask to wrap the run function, allowing the system to accept rest API requests (full code can be found on github [21]).

To get an answer for your question from the /query endpoint the question is passed in using the “q” parameter. This query is encoded using percent encoding to ensure all characters are sent including the question mark.

Besides a query endpoint, the /rate_response endpoint allows users to rate their answer as good or bad. The endpoint takes a rating parameter which can be 1 for good and -1 for bad. The original question id is also passed in. Following an answer from the /query endpoint, the user then has up to one hour to rate their answer. The rating is stored anonymously in a database together with the original question, answer and metadata from the individual modules.

A rest API works great for developers wanting to integrate the production system into their frontend such as with Gaea. However for quick testing and demo’s an API is not great as it requires users to first convert the query into the percent encoded parameter and then read the result from the json response. To solve this a simple webpage was created that allows users to interact with the chatbot in a more intuitive manner. The webpage in figure 7.6.2 is designed to run on any type of browser.

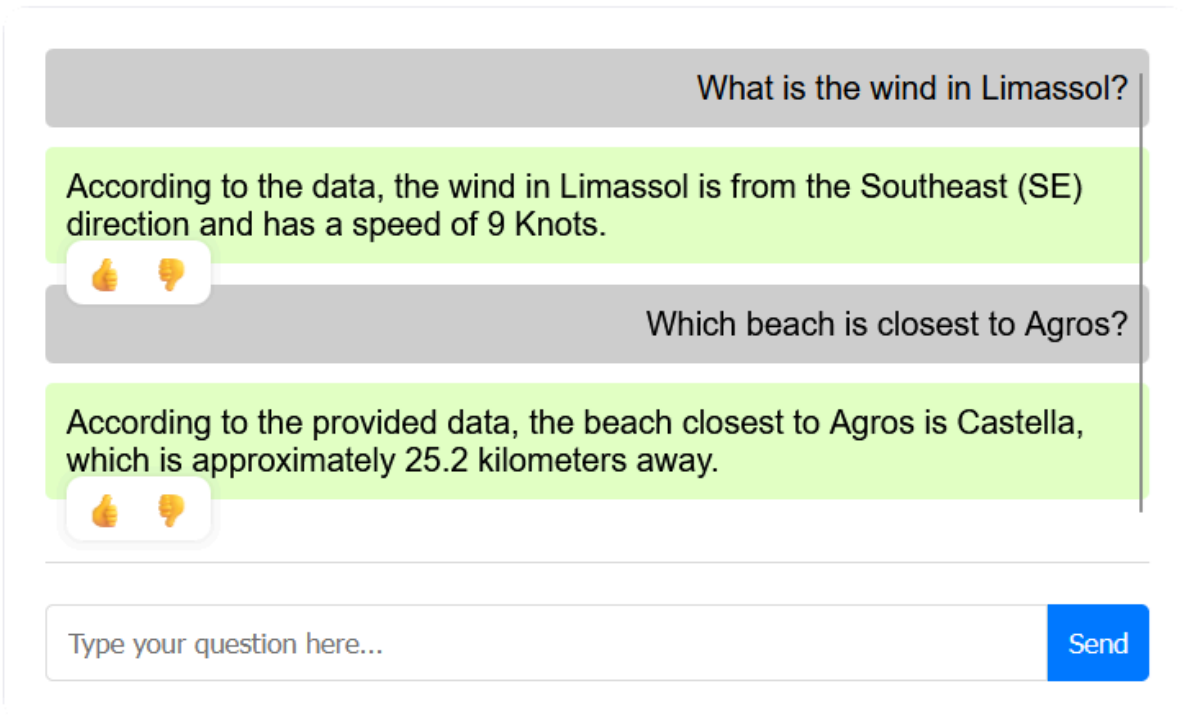
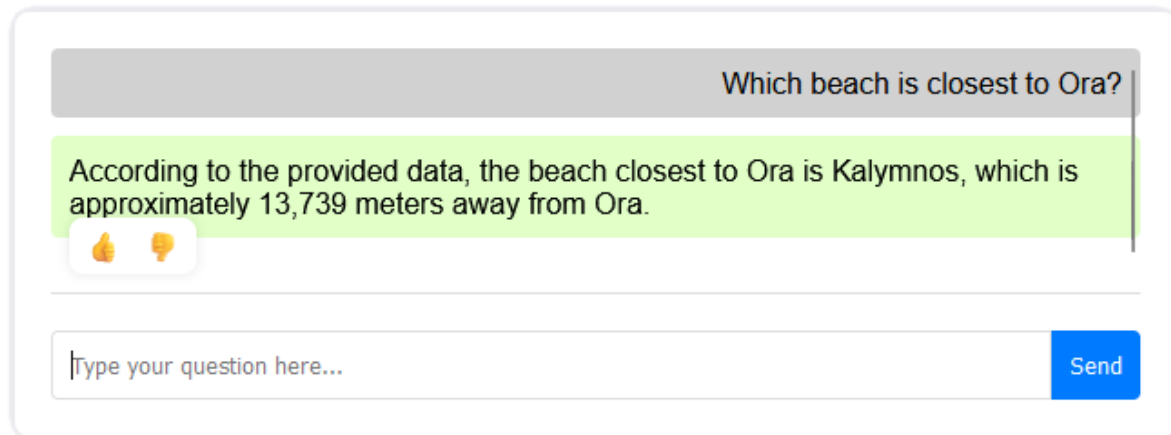


Figure 7.6.2: A simple web app that interacts with the production API and displays answers in a text box, green if the API returns success as true and red if success is false. Users can also easily rate answers using the thumbs up and thumbs down.

Chapter 8 - Evaluation

8.1 Answer evaluation

A system that uses trained models for prediction and classification is never 100% accurate. Thus users can rate their answers and provide feedback. A basic thumbs up thumbs down system is used. Using the API the resulting feedback is stored in a database allowing for later investigation and possible use for reinforcement learning in the future.



The screenshot shows a user interface for a question-answering system. At the top, a grey bar contains the question: "Which beach is closest to Ora?". Below this, a light green box displays the system's answer: "According to the provided data, the beach closest to Ora is Kalymnos, which is approximately 13,739 meters away from Ora." Underneath the answer is a rating system with two thumbs icons, one up and one down. At the bottom, there is a text input field with the placeholder "Type your question here..." and a blue "Send" button to its right.

Figure 8.1.1: Question answering text box where a user can rate their question using a thumbs up or thumbs down.

The thumbs up and thumbs down system was chosen as it allows for easy evaluation by the end user. A more complex system such as a textbox or a line scale would provide more information about the incorrect answer but would require users to put in more effort into rating their answer, resulting in less feedback.

8.2 Requirements evaluation

8.1 Accuracy

Section 5.2 describes the requirement for an accurate system. The system must have an accuracy of at least 90% when answering questions with a working API.

To evaluate the accuracy a set of questions was defined. These questions were based on questions that users had asked the system in production. Each question was asked and the answer was noted. The answers were evaluated to determine if they were correct. The results can be observed in figure 8.2.

Absolute system accuracy

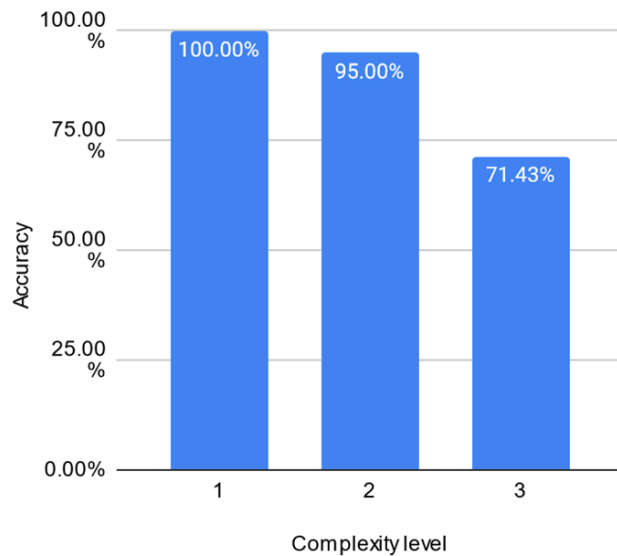


Figure 8.2: System accuracy per complexity level for the questions defined in appendix C.

In figure 8.2 showing the results of testing, a clear drop in accuracy can be observed when looking at more advanced questions such as those in complexity level 3. After investigating this drop was a result of the increasing amount of data that is being passed on into the LLM, furthermore the limited comparative functionality of an LLM also posed a limitation in question answering. Further research into LLMs could increase the answering capabilities of this system for more complex questions.

In figure 8.3, comparing the system to the prototype systems that were created in the ideation and realization phases shows us that the numerous optimizations described in chapter 7 have increased the systems answer accuracy by about twenty percent.

Average System Accuracy

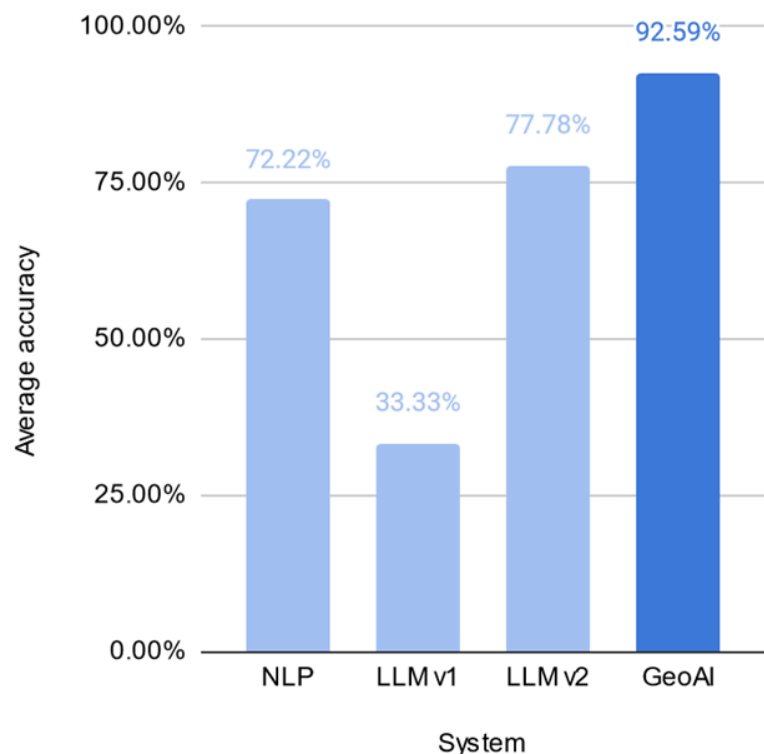


Figure 8.2: Comparing the production system (GeoAI) accuracy to the prototype system (NLP, LLM v1, LLM v2).

8.2 Speed

In section 5.2 the requirement of the system being fast was set. The system should ideally answer questions within 10 seconds to keep users focussed and motivate them to use the system more often.

To achieve this numerous optimizations were made which are described in chapter 7. These optimizations have resulted in the production system answering questions within 8.03 seconds on average. On top of that 75% of all questions are answered within 7.23 seconds.

These calculations were made based on the recorded time from a request being received by the server to the time that a response is returned to the server. Extra delays outside our control such as those created by a poor internet connection were ignored and not logged. As the server running the production application is connected to the university network through fast ethernet the network delay is negligible.

8.3 Local Deployment

Another non functional requirement was that the system must be able to be deployed locally. Data must not be shared with third parties. Using NLP tools such as Spacy and LLMs such as Llama3 allows the system to be run locally on any server that has enough resources. The only current issue preventing the system from working completely offline is the geocoding API that is provided by a third party company. In theory as time progresses all the vast amount of locations will be cached in the database, eliminating the need for the third party API. Another option would be to curate a list of all possible locations in Cyprus and use that for geocoding.

8.4 Integratability

As defined in section 5.1 and 5.2 the system must be easy to integrate for developers. Allowing the system to be implemented with as little effort as possible. To achieve this a well defined Readme.md file was created (appendix B). Clear API documentation was made and according to the developers polled it was clear to understand. As this requirement is hard to measure, feedback was requested from experienced developers. The end user, the Gaea frontend developer found the documentation very clear to understand and was able to integrate the API seamlessly.

Chapter 9 - Discussion & Future work

The field of geospatial sciences is always growing and so is this NLP system. There are many features that could be implemented to allow for a wider use of the system.

Firstly the current system always needs to make a request. The system could be modified so that when it does not detect that a question is about a specific dataset or endpoint it would instead refer the question to another LLM. This LLM could be finetuned as a help agent for Gaea, allowing users to ask questions about the Gaea tool.

The answer evaluation feature that allows users to rate their answers currently only stores the rating together with the request logs. In the future this data could well be used for manual evaluation of individual modules and seeing where modules could be improved. Ideally this process of constantly improving the system using the feedback would be done automatically. Reinforcement learning could be used to allow the LLM to learn directly from its mistakes. More research would be needed to prevent the LLM from over compensating.

Not all questions are answered correctly. The LLM responsible for answer generation is often the module where the data is not correctly used to answer the question. Advancements in the future will create a more powerful LLM. The use of Ollama in this module allows for easy substitution of LLMs.

Sometimes however the LLM does not receive correct data. This is often caused by the location detection module. As seen in section 7.2, where logic is used to correct for the difficult linguistics, the system struggles to identify locational terms consistently as Greek and Turkish linguistics are hard to understand. The solution could be the creation of a custom location detection model, it could still be based on Bert but then be trained on all the different names and locations in cyprus.

The current system only answers relatively simple questions that can be about one topic but contain multiple locations. Future research could expand on this by allowing for more complex questions to be asked and accurately answered.

Using LLMs that have been trained on tool use was researched and tested. Current open source LLMs do not yet have the power to accurately reason. Future LLMs however might have this power and allow us to implement the toolled LLM with high accuracy in a production system. Allowing for more easy scaling into answering higher complexity level questions.

Chapter 10 - Conclusion

This research aimed to build a system that allows users to interact with geospatial data using natural language. Initially a literature review was conducted to show the current state of the art and the limitations of existing solutions. Several natural language processing (NLP) and geospatial tools were analyzed and tested to get a firm understanding of the problem and an understanding of how a solution might be built. Using the knowledge gained, requirements were set in the specification phase. Using a combination of NLP and large language models (LLMs) a system was realized that allows users to ask natural language questions and get natural language answers that have been generated by referencing geospatial data to answer these questions.

This research also identified several limitations. The occasional hallucinations by LLMs that result in incorrect data and answers poses a challenge that needs to be addressed in future research. This however is a larger problem that is not specific to the geospatial implementation done in this project. Future work should focus on increasing the accuracy and reliability of LLMs in geospatial contexts. The system's performance could also improve by developing new methods to connect LLMs to datasets and by allowing for increasingly complex and at the same time general questions to be asked. The current system can accurately answer questions upto the third complexity level as defined in section 4.4. Future research could allow for more complex questions to be processed.

In summary, this research highlights the potential of NLP and LLMs in transforming natural language interactions with geospatial data. The advancement in the use of NLP and LLMs in geospatial data will increase accessibility and usability of geospatial data. Continued research and development in the fields of NLP and LLMs will allow full geospatial question answer systems to be built in the future.

References

- [1] A. Jamil, C. Padubidri, S. Karatsiolis, I. Kalita, A. Guley, and A. Kamilaris, "GAEA: A Country-Scale Geospatial Environmental Modelling Tool: Towards a Digital Twin for Real Estate," *Springer Nature Switzerland*, Jan. 01, 2024.
- [2] "PD-10 - Natural Language Processing in GIScience Applications," *GIS&T Body of Knowledge*.
<https://gistbok.ucgis.org/bok-topics/natural-language-processing-giscience-applications>
- [3] M. Cai, "Natural language processing for urban research: A systematic review," *Heliyon*, vol. 7, no. 3, p. e06322, Mar. 2021, doi: 10.1016/j.heliyon.2021.e06322.
- [4] H. Shelar, G. Kaur, N. Heda, and P. Agrawal, "Named Entity Recognition Approaches and Their Comparison for Custom NER Model," *Science & Technology Libraries*, vol. 39, no. 3, pp. 324–337, May 2020, doi: 10.1080/0194262x.2020.1759479.
- [5] E. Helderop, J. Huff, F. Morstatter, A. Grubestic, and D. Wallace, "Hidden in Plain Sight: A Machine Learning Approach for Detecting Prostitution Activity in Phoenix, Arizona," *Applied Spatial Analysis and Policy*, vol. 12, no. 4, pp. 941–963, Nov. 2018, doi: 10.1007/s12061-018-9279-1.
- [6] "Voice-based Road Navigation System Using Natural Language Processing (NLP)," *IEEE Xplore*. <https://ieeexplore.ieee.org/abstract/document/8913387>
- [7] G. Mai, K. Janowicz, C. He, S. Liu, and N. Lao, "POIReviewQA," in *Proceedings of the 12th Workshop on Geographic Information Retrieval*, Nov. 2018. Accessed: Jul. 06, 2024. [Online]. Available: <https://arxiv.org/pdf/1810.02802>
- [8] S. Muralidharan, F. Tung, and G. Mori, "PlacesQA: Towards Automatic Answering of Questions on the Web".
- [10] Z. Yin, C. Zhang, D. W. Goldberg, and S. Prasad, "An NLP-based Question Answering Framework for Spatio-Temporal Analysis and Visualization," Mar. 2019, doi: <https://doi.org/10.1145/3318236.3318240>.

- [11] “Large Language Models (LLMs),” *MongoDB*.
<https://www.mongodb.com/resources/basics/large-language-models>
- [12] Y. Jiang and C. Yang, “Is ChatGPT a Good Geospatial Data Analyst? Exploring the Integration of Natural Language into Structured...,” *MDPI*, Jan. 10, 2024.
https://www.researchgate.net/publication/377319486_Is_ChatGPT_a_Good_Geospatial_Data_Analyst_Exploring_the_Integration_of_Natural_Language_into_Structured_Query_Language_within_a_Spatial_Database
- [13] D. Punjani, S.-A. Kefaldis, K. Plas, E. Tsalapati, M. Koubarakis, and P. Maret, “The Question Answering System GeoQA2”.
- [14] Z. Li and H. Ning, “Autonomous GIS: the next-generation AI-powered GIS,” May 2023, doi:
<https://doi.org/10.48550/arxiv.2305.06453>.
- [15] A. Fernandez and S. Dube, “CORE BUILDING BLOCKS: NEXT GEN GEO SPATIAL GPT APPLICATION.” Accessed: Jul. 18, 2024. [Online]. Available:
<https://arxiv.org/ftp/arxiv/papers/2310/2310.11029.pdf>
- [16] Y. Zhang, C. Wei, S. Wu, Z. He, and W. Yu, “ARTICLE TEMPLATE GeoGPT: Understanding and Processing Geospatial Tasks through An Autonomous GPT.” Accessed: Jul. 18, 2024. [Online]. Available: <https://arxiv.org/pdf/2307.07930>
- [17] “Berkeley Function Calling Leaderboard (aka Berkeley Tool Calling Leaderboard).”
<https://gorilla.cs.berkeley.edu/leaderboard.html>
- [18] A. Mader and W. Eggink, “A DESIGN PROCESS FOR CREATIVE TECHNOLOGY,” *unknown*, Sep. 05, 2014.
https://www.researchgate.net/publication/265755092_A_DESIGN_PROCESS_FOR_CREATIVE_TECHNOLOGY
- [19] “SuPerWorld Geo-API: Artificial intelligence in real estate for modelling risks,” *SuPerWorld Research Group*. <https://superworld.cyens.org.cy/product1.html> (accessed Apr. 17, 2024).
- [20] U. Gnewuch, S. Morana, M. T. P. Adam, and A. Maedche, “Opposing Effects of Response

Time in Human–Chatbot Interaction: The Moderating Role of Prior Experience,” *Springer Nature*, May 30, 2022.

https://www.researchgate.net/publication/360950385_Opposing_Effects_of_Response_Time_in_Human-Chatbot_Interaction_The_Moderating_Role_of_Prior_Experience

[21] V. Embrechts, "Meliferea," GitHub repository, <https://github.com/darkroasted/Meliferea>, accessed July 16, 2024.

[22] "LocationIQ," *Free Reverse Geocoding API, Geocoding API, Autocomplete API*. <https://locationiq.com/> (accessed Jul. 16, 2024).

[23] "google-bert/bert-base-uncased · Hugging Face." <https://huggingface.co/google-bert/bert-base-uncased> (accessed Jul. 17, 2024).

[24] "dslim/bert-base-NER · Hugging Face." <https://huggingface.co/dslim/bert-base-NER> (accessed Jul. 17, 2024).

[25] E. F. Tjong Kim Sang and F. De Meulder, "Introduction to the CoNLL-2003 Shared Task: Language-Independent Named Entity Recognition," in **Proc. Seventh Conf. Natural Language Learning at HLT-NAACL 2003**, 2003, pp. 142-147. [Online]. Available: <https://www.aclweb.org/anthology/W03-0419>

[26] "nielsr/bert-finetuned-ner · Hugging Face." <https://huggingface.co/nielsr/bert-finetuned-ner> (accessed Jul. 17, 2024).

[27] "dslim/bert-large-NER · Hugging Face." <https://huggingface.co/dslim/bert-large-NER> (accessed Jul. 17, 2024).

[28] "Training Pipelines & Models · spaCy Usage Documentation," *Training Pipelines & Models*. <https://spacy.io/usage/training>

Appendix A - Template questions

Appendix A1 - Vicinity related Template Questions

How close is [*location*] to [*object*]
What is the distance from [*location*] to [*object*]
How far away is [*object*] from [*location*]
What is the proximity of [*object*] to [*location*]
How distant is [*object*] from [*location*]
How far is [*location*] located from [*object*]?
What is the approximate distance from [*location*] to [*object*]?
How far is [*object*] situated from [*location*]?
What is the distance between [*object*] and [*location*]
What is the distance between [*location*] and [*object*]

Table A1: Template questions relating to vicinity, containing placeholders for the location and the object of the sentence.

Appendix A2 - Questions related to weather

question	category	comp	topic	location	time
What is the temperature in Agros in May 2023?	temperature	FALSE	temperature	Agros	May 2023
How hot was it in Nicosia on June 2021?	temperature	FALSE	hot	Nicosia	June 2021
How warm was it in Lofou in January 2019?	temperature	FALSE	warm	Lofou	January 2019
What was the warmest month in Agros in 2020?	temperature	TRUE	warmest month	Agros	2020
How cold was it in Marki in February 2023?	temperature	FALSE	cold	Marki	February 2023
Which month was the coldest in Agraka in 2018?	temperature	TRUE	coldest month	Agraka	2018
How cold was the coldest month in Paphos?	temperature	TRUE	coldest month	Paphos	
In the Nicosia district, which town was the hottest in september 2023?	temperature	TRUE	hottest location	Nicosia district	September 2023
What was the overall coldest year for Peyia?	temperature	TRUE	coldest year	Peyia	
For how many months a year is the average temperature under 10°C in Phicardou?	temperature	TRUE	temperature under 10*c	Phicardou	
Which year had the hottest June in Zygi?	temperature	TRUE	hottest year	Zygi	June
What was the highest temperature recorded during the summer of 2016?	temperature	TRUE	highest temperature		summer of 2016
What was the hottest place during the winter of 2021-2022?	temperature	TRUE	hottest location		winter 2021-2022
How humid was it in Paphos forest on March 2021?	humidity	FALSE	humidity	Paphos Forest	March 2021
What was the humidity level in Pano Lefkara in April 2018?	humidity	FALSE	humidity	Pano Lefkara	April 2018
Is the summer or winter more humid on Girne?	humidity	TRUE	humidity season	Girne	
What month was the most humid in Dali in 2022?	humidity	TRUE	humidity month	Dali	2022
In Kapulica, which month is the most humid?	humidity	TRUE	humidity month	Kapulica	

Which city is the most humid of the Famagusta district during october?	humidity	TRUE	humidity location	Famagusta	october
Which year had the most humid July in Aloa?	humidity	TRUE	humidity	Aloa	July
Which month is more humid in Omodos? January or July?	humidity	TRUE	humidity	Omodos	January, July
How dry was Pomos in August 2021?	humidity	FALSE	dryness	Pomos	August 2021
Which year had the dampest August in Pissouri?	humidity	TRUE	damp	Pissouri	August
When was the driest month recorded in Limassol?	humidity	TRUE	driest month	Limassol	
How dry was the driest month in Astrometritis?	humidity	TRUE	driest month	Astrometritis	
Where was it the driest in November 2017?	humidity	TRUE	driest location		November 2017
How damp was it in Peyia during December 2019?	humidity	FALSE	dampness	Peyia	December 2019
How moist was it in Pissouri in March 2018?	humidity	FALSE	moistness	Pissouri	March 2018
How windy was it in Germansoigia?	wind	FALSE	windiest	Germansoigia	
Which month was the windiest in Kalavassos?	wind	TRUE	windiest month	Kalavassos	
How windy was the windiest month in Mazotos?	wind	TRUE	windiest month	Mazotos	
How windy is Kirkklar on average each year?	wind	TRUE	windy	Kirkklar	
What is the windiest city in Cyprus?	wind	TRUE	windiest location	Cyprus	
How fast was wind in the windiest month in Pasakoy?	wind	TRUE	windiest month	Pasakoy	
How fast was the wind in Bafra during	wind	FALSE	wind speed	Bafra	
From where did the wind come in Kirkklar?	wind	FALSE	wind direction	Kirkklar	
What was the most prominent wind direction in Liopetri?	wind	FALSE	wind direction	Liopetri	
What is the average yearly wind speed in Bahceli?	wind	FALSE	wind speed	Bahceli	

Which city has the fastest wind yearly?	wind	TRUE	wind speed		
What was the wind speed on Mount Olympos?	wind	FALSE	wind speed	Mount Olympos	
How fast was the wind on average in Pyrgos?	wind	FALSE	wind speed	Pyrgos	
Which mountain has the fastest wind?	wind	TRUE	wind speed	mountain	
Where on the island can i find the fastest wind?	wind	TRUE	wind speed	island	

Table A2: Sample questions about the weather that have been annotated.

Appendix A3 - Weather related template questions

How hot was it in [*location*] on [*date*]?	temperature	hot
What was the temperature on [*date*] in [*location*]?	temperature	temperature
How warm was it in [*location*] on [*date*]?	temperature	warm
What was the hottest month in [*location*]?	temperature	hottest
How cold was it in [*location*] on [*date*]?	temperature	cold
When was the coldest month in [*location*]?	temperature	coldest
Where was it the coldest in [*date*]	temperature	coldest
How fast is the wind in [*location*] on average?	wind	wind
At what speed does the wind in [*location*] blow on average?	wind	wind
From what direction does the wind come from in [*location*]?	wind	wind
What is the prominent wind direction in [*location*]?	wind	wind
Where does the wind come from in [*location*]?	wind	wind
What was on average the most humid month in [*location*]?	humidity	most humid
How humid was it in [*location*] during [*month*]?	humidity	humid
What was the highest recorded humidity in [*location*]	humidity	humidity
Which month has the lowest humidity in [*location*]	humidity	lowest humidity
What was the precipitation like in [*location*] during [*month*]?	precipitation	precipitation
Which month had the highest precipitation in [*location*]?	precipitation	highest precipitation
What was the rainfall score in [*location*] during [*month*]?	precipitation	rainfall score
How much rainfall is there on average in [*location*] during the year?	precipitation	rainfall
What is the average rainfall in [*month*] in [*location*]?	precipitation	average rainfall
Which month has the least rainfall in [*location*]?	precipitation	least rainfall

Table A3: Template question about the weather per endpoint.

Appendix A4 - Template questions relating to vicinity

How close is [*location*] to [*object*]
What is the distance from [*location*] to [*object*]
How far away is [*object*] from [*location*]
What is the proximity of [*object*] to [*location*]
How distant is [*object*] from [*location*]
How far is [*location*] located from [*object*]?
What is the approximate distance from [*location*] to [*object*]?
How far is [*object*] situated from [*location*]?
What is the distance between [*object*] and [*location*]
What is the distance between [*location*] and [*object*]

Table A4: template questions relating to the vicinity domain, containing placeholders for the object and the location.

Appendix B - API Documentation

Note this is a copy of the api documentation that can be found in the github repository [21]

There are two api endpoints:

1. **/query** for asking questions and getting an answer from the system
2. **/rate_response** for rating the response given

/query endpoint

To get an answer for your question you can use the **/query** endpoint with a **q** parameter that is set to the query that the user has. This query is encoded using **percent encoding** to ensure all characters are sent.

A request will almost always return 3 values:

- **"success"**: A boolean that when everything went correctly is set to **True** and when an error occurred is set to **False**
- **"answer"**: provides the system answer when **"success": True**, else it provides a hint as to what went wrong and sometimes what the user may do to improve the chance their query succeeds
- **"request_id"**: A unique ID for the request that can be used to rate the request (See **/rate_response** endpoint)

URL: [GET] `http://c53.student.utwente.nl/1X85B95UQGNI/query?q={INSERT ENCODED QUESTION}`

Example

For example the question **“What is the average temperature in Limassol?”** would initiate this request:

```
[GET]http://c53.student.utwente.nl/1X85B95UQGNI/query?q=How+warm+is+it+in+Limassol%3F
```

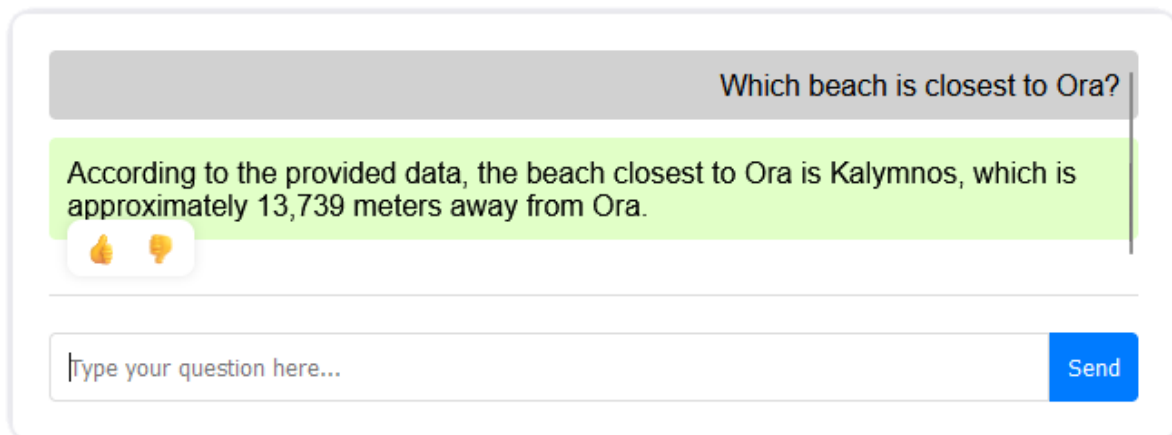
This request would then return the following code after about 10 seconds:

```
{
  "request_id": 178327,
  "success": true,
  "answer": "According to the provided temperature data for Limassol, the average temperature throughout the year ranges from around 12.6°C in
```

```
January (the coolest month) to approximately 29.2°C in July and August (the warmest months). So, it's generally quite warm in Limassol during the summer season."
```

```
}
```

/rate_response endpoint



The screenshot shows a chat interface. At the top, a grey bar contains the question: "Which beach is closest to Ora?". Below this, a light green box contains the answer: "According to the provided data, the beach closest to Ora is Kalymnos, which is approximately 13,739 meters away from Ora." Underneath the answer, there are two thumbs-up icons and one thumbs-down icon, indicating a rating system. At the bottom, there is a text input field with the placeholder "Type your question here..." and a blue "Send" button.

To allow the system to learn, users can provide feedback about their request. They can give it a 👍 or 👎 to rate the answer they got. The frontend must translate this into a request to the `/rate_response` endpoint.

The `/rate_response` endpoint takes 2 parameters:

- `request_id`: is the original request id of the request that we want to rate
- `rating`: can be 1 (positive) or -1 (negative). So 👍 -> 1 and 👎 -> -1.

To prevent misuse feedback can only be provided up to 1 hour after the original answer from the `/query` has been given!

URL:

```
[GET]http://c53.student.utwente.nl/1X85B95UQGNI/rate_response?request_id={INSERT REQUEST ID}&rating={INSERT RATING}
```

Example

If we want to rate the answer that we got in the `/query` example as correct we can initiate the following request:

[GET]

`http://c53.student.utwente.nl/1X85B95UQGNI/rate_response?request_id=178327&rating=1`

This can return 2 responses:

success: **true**

```
{
  "success":true,
  "error":"no errors :)"
}
```

-

success: **false**

```
{
  "error": "Something went wrong but we cant specifically say what :( ",
  "success": false
}
```

At times when we know what went wrong the system may provide a hint as to how it can be fixed but sometimes it will return this. When a non descript error like `Something went wrong but we cant specifically say what :(` occurs this is mostly because the timeout limit of 1 hour has been crossed.

Appendix C - Evaluation questions

Questions and relevant data pairs. Used in section 7.5.

Question	Data
How close is Aipeias Ave in Nicosia to the sea?	[{"Data on distance-sea for Aipeias Ave: {'Coordinates': [35.33482250038557, 33.34833150013221], 'Distance': 19673.787893804485, 'Unit': 'm'}", "Data on distance-sea for Nicosia: {'Coordinates': [35.3338159996877, 33.37023980013088], 'Distance': 17668.16369105547, 'Unit': 'm'}"]
How much risk is there of a wildfire at Agiou Andreou 3035?	[{"Data on wildfire-risk for Agiou Andreou: {'Coordinates': [35.17255877609483, 33.34723530193588], 'Distance': 26.58331186722666, 'Risk': 'Very Low Risk', 'Score': 1, 'Unit': 'm'}"]
Please give me information about the vegetation at Famagusta	[{"Data on vegetation for Famagusta: {'Vegetation': 'no vegetation', 'Vegetation Score': 0}"]
Where does the wind blow to in Melini?	[{"Data on wind for Melini: {'Unit': 'Knots', 'Wind Direction': 'SE', 'Wind Speed': 10}"]
How likely is a flood at Oron street Larnaca?	[{"Data on flooding-risk for Larnaca: {'Coordinates': [34.9236095, 33.6236184], 'Risk': 'Very Low Risk', 'Score': 1}"]
What is the slope at Digeni Ave 5281?	[{"Data on aspect-slope for Digeni Ave: {'Aspect': 1, 'Coordinates': [35.1706277, 33.3759634], 'Slope': 7, 'Unit': 'Degree'}"]
Is Limassol in a natura 2000 location?	[{"Data on natura-region for Limassol: {'Closest Zone Index': 39, 'Distance from Natura': 7924.347578318594, 'Natura_Region': 0, 'Unit': 'meters'}"]
What is the vegetation like at 20 Nikis Ave in Nicosia?	[{"Data on vegetation for Nikis Ave: {'Vegetation': 'no vegetation', 'Vegetation Score': 0}", "Data on vegetation for Nicosia: {'Vegetation': 'no vegetation', 'Vegetation Score': 0}"]
What is the elavation at Trodos?	[{"Data on aspect-slope for Trodos: {'Aspect': 195, 'Coordinates': [34.6980313, 33.0175999], 'Slope': 2, 'Unit': 'Degree'}"]

Provide me all temperature data for Xenopoulou Limassol	["Data on temperature for Limassol: [{'Average': 12.626734693877552, 'Minimum': 12.535714285714286, 'Maximum': 12.717755102040815, 'Unit': 'Degree Celsius', 'month': 1}, {'Average': 14.662893715479921, 'Minimum': 13.609770114942528, 'Maximum': 15.716017316017316, 'Unit': 'Degree Celsius', 'month': 2}, {'Average': 16.528360215053766, 'Minimum': 16.011021505376345, 'Maximum': 17.045698924731184, 'Unit': 'Degree Celsius', 'month': 3}, {'Average': 19.834236111111111, 'Minimum': 18.680694444444445, 'Maximum': 20.987777777777778, 'Unit': 'Degree Celsius', 'month': 4}, {'Average': 23.19758064516129, 'Minimum': 22.64354838709677, 'Maximum': 23.75161290322581, 'Unit': 'Degree Celsius', 'month': 5}, {'Average': 26.27835371634433, 'Minimum': 24.863651877133105, 'Maximum': 27.693055555555556, 'Unit': 'Degree Celsius', 'month': 6}, {'Average': 29.294339622641512, 'Minimum': 29.294339622641512, 'Maximum': 29.294339622641512, 'Unit': 'Degree Celsius', 'month': 7}, {'Average': 29.24099462365591, 'Minimum': 29.24099462365591, 'Maximum': 29.24099462365591, 'Unit': 'Degree Celsius', 'month': 8}, {'Average': 26.930434782608696, 'Minimum': 26.930434782608696, 'Maximum': 26.930434782608696, 'Unit': 'Degree Celsius', 'month': 9}, {'Average': 24.28266129032258, 'Minimum': 24.28266129032258, 'Maximum': 24.28266129032258, 'Unit': 'Degree Celsius', 'month': 10}, {'Average': 18.996105702364396, 'Minimum': 18.996105702364396, 'Maximum': 18.996105702364396, 'Unit': 'Degree Celsius', 'month': 11}, {'Average': 12.854504504504504, 'Minimum': 12.854504504504504, 'Maximum': 12.854504504504504, 'Unit': 'Degree Celsius', 'month': 12}]]"]
What is the average temperature in Makariou Avenue Limassol in August?	["Data on temperature for Makariou Avenue Limassol: [{'Average': 11.804502688172043, 'Minimum': 11.225806451612904, 'Maximum': 12.383198924731182, 'Unit': 'Degree Celsius', 'month': 1}, {'Average': 13.689186427680587, 'Minimum': 12.04640804597701, 'Maximum': 15.331964809384164, 'Unit': 'Degree Celsius', 'month': 2}, {'Average': 15.274327956989247, 'Minimum': 14.17016129032258, 'Maximum': 16.378494623655914, 'Unit': 'Degree Celsius', 'month': 3}, {'Average': 18.420555555555556, 'Minimum': 16.748055555555556, 'Maximum': 20.093055555555555, 'Unit': 'Degree Celsius', 'month': 4}, {'Average': 21.76801075268817, 'Minimum': 21.766532258064515, 'Maximum': 21.769489247311828, 'Unit': 'Degree Celsius', 'month': 5}, {'Average': 24.765972222222224, 'Minimum': 23.052777777777777, 'Maximum': 26.479166666666668, 'Unit': 'Degree Celsius', 'month': 6}, {'Average': 27.39286132939695, 'Minimum': 26.608064516129033, 'Maximum': 28.17765814266487, 'Unit': 'Degree Celsius', 'month': 7}, {'Average': 27.568212365591396, 'Minimum': 26.914112903225803, 'Maximum': 28.22231182795699, 'Unit': 'Degree Celsius', 'month': 8}, {'Average': 26.309652777777778, 'Minimum': 26.052638888888889, 'Maximum': 26.566666666666666, 'Unit': 'Degree Celsius', 'month': 9}, {'Average': 23.25257159315339, 'Minimum': 23.079973118279568, 'Maximum': 23.42517006802721, 'Unit': 'Degree Celsius', 'month': 10}, {'Average': 18.1175, 'Minimum': 17.813055555555554, 'Maximum': 18.421944444444442, 'Unit': 'Degree Celsius', 'month': 11}, {'Average': 13.706586021505377, 'Minimum': 12.544354838709678, 'Maximum': 14.868817204301076, 'Unit': 'Degree Celsius', 'month': 12}]]"]
How far is Antoni Samaraki 3117 from a road?	["Data on distance-road for Antoni Samaraki: [{'Coordinates': [34.712670489888175, 33.031387513618526], 'Distance': 0.17943243625346322, 'Type': 'residential', 'Unit': 'm'}, {'Coordinates': [34.71265686713147, 33.03138607453055], 'Distance': 1.6964108271612586, 'Type': 'proposed', 'Unit': 'm'}, {'Coordinates': [34.71231710034305, 33.03133309986523], 'Distance': 39.69835187592883, 'Type': 'residential', 'Unit': 'm'}, {'Coordinates': [34.71316954114658, 33.031453503536135], 'Distance': 55.51205325970738, 'Type': 'service', 'Unit': 'm'}, {'Coordinates': [34.712130042472815, 33.03132966843828], 'Distance': 60.367849233662284, 'Type': 'residential', 'Unit': 'm'}, {'Coordinates': [34.713203300563926, 33.03119829957202], 'Distance': 61.4300022129353, 'Type': 'service', 'Unit': 'm'}, {'Coordinates': [34.71307300064087, 'Distance': 61.4300022129353, 'Type': 'service', 'Unit': 'm'}]]"]

	33.03218330048441], 'Distance': 85.37964064677227, 'Type': 'service', 'Unit': 'm'}, {'Coordinates': [34.712128900470226, 33.032185799597535], 'Distance': 94.7443659926427, 'Type': 'residential', 'Unit': 'm'}, {'Coordinates': [34.712252299727325, 33.03045190018224], 'Distance': 97.5583926380372, 'Type': 'residential', 'Unit': 'm'}, {'Coordinates': [34.71286281056608, 33.03022530411089], 'Distance': 108.56386934024468, 'Type': 'residential', 'Unit': 'm'}, {'Coordinates': [34.713668466790324, 33.031518535290566], 'Distance': 111.18017427183122, 'Type': 'service', 'Unit': 'm'}, {'Coordinates': [34.71185616232489, 33.0323677737758], 'Distance': 127.4906485598522, 'Type': 'residential', 'Unit': 'm'}, {'Coordinates': [34.71161440003438, 33.032166500192], 'Distance': 137.32323761064276, 'Type': 'residential', 'Unit': 'm'}, {'Coordinates': [34.71254229969941, 33.032962400347195], 'Distance': 144.96921342498146, 'Type': 'residential', 'Unit': 'm'}, {'Coordinates': [34.71254229969941, 33.032962400347195], 'Distance': 144.96921342498146, 'Type': 'residential', 'Unit': 'm'}, {'Coordinates': [34.71231769995192, 33.02966449988624], 'Distance': 162.67830534070671, 'Type': 'residential', 'Unit': 'm'}, {'Coordinates': [34.712135699991265, 33.03322630022484], 'Distance': 178.6302912063912, 'Type': 'residential', 'Unit': 'm'}, {'Coordinates': [34.71313890013175, 33.03327709995417], 'Distance': 180.66094329205396, 'Type': 'residential', 'Unit': 'm'}, {'Coordinates': [34.711087216247, 33.030795702532714], 'Distance': 183.9932989908609, 'Type': 'residential', 'Unit': 'm'}, {'Coordinates': [34.71416370012791, 33.030444599573926], 'Distance': 186.66669798020015, 'Type': 'residential', 'Unit': 'm'}}"]]
Which beach is closest to Agros?	[["Data on nearest-blueflag-beach for Agros: [{"Beach Coordinates": [34.70403512909493, 33.11212107810617], 'Distance': 25200.48169457493, 'Municipality': 'Agios Tychonas Community board', 'Name': 'Castella', 'Unit': 'm'}]]"]]
Is an earthquake likely to happen at 19 Afroditis Nicosia?	[["Data on seismic-Zone for Nicosia: None"]]
How risky is a landslide at Demostheni Severi Avenue?	[["Data on landslides-risk for Demostheni Severi Avenue: {'Coordinates': [35.164496799700636, 33.354070223844005], 'Distance': 97.89247793220835, 'Risk': 'Very Low Risk', 'Score': 1, 'Unit': 'm'}"]]
How fast is the wind in Andrea Zakou 7101?	[["Data on wind for Andrea Zakou: {'Unit': 'Knots', 'Wind Direction': 'SE', 'Wind Speed': 8}"]]
What type of road is closest to Lefke?	[["Data on distance-road for Lefke: []"]]
What type of winds can I expect at mount Olympus?	[["Data on wind for Olympus: {'Unit': 'Knots', 'Wind Direction': 'SE', 'Wind Speed': 12}"]]

<p>How many trees are there around Larnakos Ave 2101 ?</p>	<p>["Data on near-tree for Larnakos Ave: {'Trees': [{'Coordinates': [35.17254438664845, 33.37212480111008], 'Distance': 11.495980625613468, 'Tree': 1}, {'Coordinates': [35.17257082828566, 33.371923105440544], 'Distance': 14.925076061647559, 'Tree': 2}, {'Coordinates': [35.17258171601863, 33.37202395327532], 'Distance': 7.170711393293223, 'Tree': 3}, {'Coordinates': [35.172535054305904, 33.37202395327532], 'Distance': 11.866039024633997, 'Tree': 4}, {'Coordinates': [35.172594159142015, 33.37237977488101], 'Distance': 28.934720476680948, 'Tree': 5}, {'Coordinates': [35.172594159142015, 33.371873632917826], 'Distance': 18.160381162682505, 'Tree': 6}, {'Coordinates': [35.17280102606843, 33.37183367434179], 'Distance': 27.99039668805683, 'Tree': 7}, {'Coordinates': [35.17283057848649, 33.371869827339154], 'Distance': 28.024735631544583, 'Tree': 8}], 'Unit': 'm'}"]</p>
<p>Is Damagitou street or Rossias street closer to a beach?</p>	<p>["Data on nearest-blueflag-beach for Damagitou: [{'Beach Coordinates': [34.681300000173415, 33.0548726002827], 'Distance': 3498.4914004809343, 'Municipality': 'Limassol Municipality', 'Name': 'Akti Olympion A', 'Unit': 'm'}]", "Data on nearest-blueflag-beach for Rossias: [{'Beach Coordinates': [34.681300000173415, 33.0548726002827], 'Distance': 6088.898811744518, 'Municipality': 'Limassol Municipality', 'Name': 'Akti Olympion A', 'Unit': 'm'}"]]</p>
<p>When is it the rainiest in Kifisias 4549?</p>	<p>["Data on precipitation for Kyprou: [{'Average': 130.428, 'Month': 1, 'Score': 'high', 'Unit': 'mm'}, {'Average': 77.0578, 'Month': 2, 'Score': 'moderate', 'Unit': 'mm'}, {'Average': 64.9118, 'Month': 3, 'Score': 'moderate', 'Unit': 'mm'}, {'Average': 24.0919, 'Month': 4, 'Score': 'low', 'Unit': 'mm'}, {'Average': 44.7474, 'Month': 5, 'Score': 'low', 'Unit': 'mm'}, {'Average': 8.29263, 'Month': 6, 'Score': 'low', 'Unit': 'mm'}, {'Average': 2.32986, 'Month': 7, 'Score': 'low', 'Unit': 'mm'}, {'Average': 2.67441, 'Month': 8, 'Score': 'low', 'Unit': 'mm'}, {'Average': 3.97256, 'Month': 9, 'Score': 'low', 'Unit': 'mm'}, {'Average': 19.3452, 'Month': 10, 'Score': 'low', 'Unit': 'mm'}, {'Average': 34.4878, 'Month': 11, 'Score': 'low', 'Unit': 'mm'}, {'Average': 100.317, 'Month': 12, 'Score': 'high', 'Unit': 'mm'}"]]</p>
<p>Does it get colder in Paphos or Limassol?</p>	<p>["Data on temperature for Paphos: [{'Average': 12.436962365591398, 'Minimum': 12.436962365591398, 'Maximum': 12.436962365591398, 'Unit': 'Degree Celsius', 'month': 1}, {'Average': 15.57413793103448, 'Minimum': 15.574137931034482, 'Maximum': 15.574137931034482, 'Unit': 'Degree Celsius', 'month': 2}, {'Average': 16.622311827956988, 'Minimum': 16.622311827956988, 'Maximum': 16.622311827956988, 'Unit': 'Degree Celsius', 'month': 3}, {'Average': 19.921388888888888, 'Minimum': 19.921388888888888, 'Maximum': 19.921388888888888, 'Unit': 'Degree Celsius', 'month': 4}, {'Average': 21.325940860215052, 'Minimum': 21.325940860215052, 'Maximum': 21.325940860215052, 'Unit': 'Degree Celsius', 'month': 5}, {'Average': 26.204444444444444, 'Minimum': 26.204444444444444, 'Maximum': 26.204444444444444, 'Unit': 'Degree Celsius', 'month': 6}, {'Average': 27.697983870967743, 'Minimum': 27.697983870967743, 'Maximum': 27.697983870967743, 'Unit': 'Degree Celsius', 'month': 7}, {'Average': 27.646370967741934, 'Minimum': 27.646370967741934, 'Maximum': 27.646370967741934, 'Unit': 'Degree Celsius', 'month': 8}, {'Average': 25.624861111111112, 'Minimum': 25.624861111111111, 'Maximum': 25.624861111111111, 'Unit': 'Degree Celsius', 'month': 9}, {'Average': 23.09529569892473, 'Minimum': 23.09529569892473, 'Maximum': 23.09529569892473, 'Unit': 'Degree Celsius', 'month': 10}, {'Average': 18.629861111111111, 'Minimum': 18.629861111111111, 'Maximum': 18.629861111111111, 'Unit': 'Degree Celsius', 'month': 11}, {'Average': 13.482281059063135, 'Minimum': 13.482281059063137, 'Maximum': 13.482281059063137, 'Unit': 'Degree Celsius', 'month': 12}]", "Data on temperature for Limassol: [{'Average': 12.626734693877552, 'Minimum': 12.535714285714286, 'Maximum': 12.717755102040815, 'Unit': 'Degree Celsius', 'month': 1}, {'Average': 14.662893715479921, 'Minimum': 13.609770114942528, 'Maximum': 15.716017316017316, 'Unit': 'Degree Celsius', 'month': 2}, {'Average': 16.528360215053766, 'Minimum': 16.011021505376345, 'Maximum': 17.045698924731184, 'Unit': 'Degree Celsius', 'month': 3}, {'Average': 18.629861111111111, 'Minimum': 18.629861111111111, 'Maximum': 18.629861111111111, 'Unit': 'Degree Celsius', 'month': 4}, {'Average': 21.325940860215052, 'Minimum': 21.325940860215052, 'Maximum': 21.325940860215052, 'Unit': 'Degree Celsius', 'month': 5}, {'Average': 26.204444444444444, 'Minimum': 26.204444444444444, 'Maximum': 26.204444444444444, 'Unit': 'Degree Celsius', 'month': 6}, {'Average': 27.697983870967743, 'Minimum': 27.697983870967743, 'Maximum': 27.697983870967743, 'Unit': 'Degree Celsius', 'month': 7}, {'Average': 27.646370967741934, 'Minimum': 27.646370967741934, 'Maximum': 27.646370967741934, 'Unit': 'Degree Celsius', 'month': 8}, {'Average': 25.624861111111112, 'Minimum': 25.624861111111111, 'Maximum': 25.624861111111111, 'Unit': 'Degree Celsius', 'month': 9}, {'Average': 23.09529569892473, 'Minimum': 23.09529569892473, 'Maximum': 23.09529569892473, 'Unit': 'Degree Celsius', 'month': 10}, {'Average': 18.629861111111111, 'Minimum': 18.629861111111111, 'Maximum': 18.629861111111111, 'Unit': 'Degree Celsius', 'month': 11}, {'Average': 13.482281059063135, 'Minimum': 13.482281059063137, 'Maximum': 13.482281059063137, 'Unit': 'Degree Celsius', 'month': 12}"]]</p>

	<p>'Degree Celsius', 'month': 3}, {'Average': 19.83423611111111, 'Minimum': 18.680694444444445, 'Maximum': 20.987777777777778, 'Unit': 'Degree Celsius', 'month': 4}, {'Average': 23.19758064516129, 'Minimum': 22.64354838709677, 'Maximum': 23.75161290322581, 'Unit': 'Degree Celsius', 'month': 5}, {'Average': 26.27835371634433, 'Minimum': 24.863651877133105, 'Maximum': 27.693055555555556, 'Unit': 'Degree Celsius', 'month': 6}, {'Average': 29.294339622641512, 'Minimum': 29.294339622641512, 'Maximum': 29.294339622641512, 'Unit': 'Degree Celsius', 'month': 7}, {'Average': 29.24099462365591, 'Minimum': 29.24099462365591, 'Maximum': 29.24099462365591, 'Unit': 'Degree Celsius', 'month': 8}, {'Average': 26.930434782608696, 'Minimum': 26.930434782608696, 'Maximum': 26.930434782608696, 'Unit': 'Degree Celsius', 'month': 9}, {'Average': 24.28266129032258, 'Minimum': 24.28266129032258, 'Maximum': 24.28266129032258, 'Unit': 'Degree Celsius', 'month': 10}, {'Average': 18.996105702364396, 'Minimum': 18.996105702364396, 'Maximum': 18.996105702364396, 'Unit': 'Degree Celsius', 'month': 11}, {'Average': 12.854504504504504, 'Minimum': 12.854504504504504, 'Maximum': 12.854504504504504, 'Unit': 'Degree Celsius', 'month': 12}]]]</p>
<p>Is august warmer in Zygi or in Paphos?</p>	<p>["Data on temperature for Zygi: [{'Average': 10.826075268817204, 'Minimum': 10.56518817204301, 'Maximum': 11.086962365591399, 'Unit': 'Degree Celsius', 'month': 1}, {'Average': 12.861350574712644, 'Minimum': 11.838649425287358, 'Maximum': 13.88405172413793, 'Unit': 'Degree Celsius', 'month': 2}, {'Average': 14.679032258064517, 'Minimum': 14.135483870967741, 'Maximum': 15.22258064516129, 'Unit': 'Degree Celsius', 'month': 3}, {'Average': 17.640625, 'Minimum': 16.639861111111113, 'Maximum': 18.641388888888887, 'Unit': 'Degree Celsius', 'month': 4}, {'Average': 21.3442876344086, 'Minimum': 20.495967741935484, 'Maximum': 22.19260752688172, 'Unit': 'Degree Celsius', 'month': 5}, {'Average': 24.237916666666663, 'Minimum': 23.066388888888888, 'Maximum': 25.409444444444443, 'Unit': 'Degree Celsius', 'month': 6}, {'Average': 26.901344086021506, 'Minimum': 26.699596774193548, 'Maximum': 27.103091397849465, 'Unit': 'Degree Celsius', 'month': 7}, {'Average': 26.97056451612903, 'Minimum': 26.939919354838707, 'Maximum': 27.001209677419357, 'Unit': 'Degree Celsius', 'month': 8}, {'Average': 25.5638118723099, 'Minimum': 24.72375, 'Maximum': 26.403873744619798, 'Unit': 'Degree Celsius', 'month': 9}, {'Average': 22.730309139784946, 'Minimum': 22.18051075268817, 'Maximum': 23.28010752688172, 'Unit': 'Degree Celsius', 'month': 10}, {'Average': 17.576736111111111, 'Minimum': 17.311666666666667, 'Maximum': 17.841805555555556, 'Unit': 'Degree Celsius', 'month': 11}, {'Average': 12.813508064516128, 'Minimum': 10.908736559139786, 'Maximum': 14.718279569892472, 'Unit': 'Degree Celsius', 'month': 12}]]", "Data on temperature for Paphos: [{'Average': 12.436962365591398, 'Minimum': 12.436962365591398, 'Maximum': 12.436962365591398, 'Unit': 'Degree Celsius', 'month': 1}, {'Average': 15.57413793103448, 'Minimum': 15.574137931034482, 'Maximum': 15.574137931034482, 'Unit': 'Degree Celsius', 'month': 2}, {'Average': 16.622311827956988, 'Minimum': 16.622311827956988, 'Maximum': 16.622311827956988, 'Unit': 'Degree Celsius', 'month': 3}, {'Average': 19.921388888888888, 'Minimum': 19.921388888888888, 'Maximum': 19.921388888888888, 'Unit': 'Degree Celsius', 'month': 4}, {'Average': 21.325940860215052, 'Minimum': 21.325940860215052, 'Maximum': 21.325940860215052, 'Unit': 'Degree Celsius', 'month': 5}, {'Average': 26.204444444444444, 'Minimum': 26.204444444444444, 'Maximum': 26.204444444444444, 'Unit': 'Degree Celsius', 'month': 6}, {'Average': 27.697983870967743, 'Minimum': 27.697983870967743, 'Maximum': 27.697983870967743, 'Unit': 'Degree Celsius', 'month': 7}, {'Average': 27.646370967741934, 'Minimum': 27.646370967741934, 'Maximum': 27.646370967741934, 'Unit': 'Degree Celsius', 'month': 8}, {'Average': 25.624861111111112, 'Minimum': 25.624861111111111, 'Maximum': 25.624861111111111, 'Unit': 'Degree Celsius', 'month': 9}, {'Average': 23.09529569892473, 'Minimum': 23.09529569892473, 'Maximum': 23.09529569892473, 'Unit': 'Degree Celsius', 'month': 10}, {'Average': 18.629861111111111, 'Minimum': 18.629861111111111, 'Maximum': 18.629861111111111, 'Unit': 'Degree Celsius', 'month': 11}, {'Average': 18.629861111111111, 'Minimum': 18.629861111111111, 'Maximum': 18.629861111111111, 'Unit': 'Degree Celsius', 'month': 12}]]]</p>

	Celsius', 'month': 11}, {'Average': 13.482281059063135, 'Minimum': 13.482281059063137, 'Maximum': 13.482281059063137, 'Unit': 'Degree Celsius', 'month': 12}]]"]
When is the warmest period in Afroditis Nicosia?	[["Data on temperature for Nicosia: [{'Average': 10.365524193548389, 'Minimum': 10.191263440860215, 'Maximum': 10.53978494623656, 'Unit': 'Degree Celsius', 'month': 1}, {'Average': 12.876724137931035, 'Minimum': 11.392528735632183, 'Maximum': 14.360919540229887, 'Unit': 'Degree Celsius', 'month': 2}, {'Average': 14.894758064516129, 'Minimum': 14.366666666666665, 'Maximum': 15.422849462365592, 'Unit': 'Degree Celsius', 'month': 3}, {'Average': 19.016736111111111, 'Minimum': 17.199861111111111, 'Maximum': 20.833611111111111, 'Unit': 'Degree Celsius', 'month': 4}, {'Average': 22.928091397849464, 'Minimum': 22.41962365591398, 'Maximum': 23.436559139784944, 'Unit': 'Degree Celsius', 'month': 5}, {'Average': 27.05375, 'Minimum': 25.775555555555556, 'Maximum': 28.331944444444446, 'Unit': 'Degree Celsius', 'month': 6}, {'Average': 30.338239247311826, 'Minimum': 30.168010752688172, 'Maximum': 30.508467741935483, 'Unit': 'Degree Celsius', 'month': 7}, {'Average': 29.9559811827957, 'Minimum': 29.88467741935484, 'Maximum': 30.02728494623656, 'Unit': 'Degree Celsius', 'month': 8}, {'Average': 27.549128334960145, 'Minimum': 26.445694444444445, 'Maximum': 28.652562225475844, 'Unit': 'Degree Celsius', 'month': 9}, {'Average': 23.562567204301075, 'Minimum': 23.001747311827955, 'Maximum': 24.12338709677419, 'Unit': 'Degree Celsius', 'month': 10}, {'Average': 16.663194444444443, 'Minimum': 16.319583333333334, 'Maximum': 17.006805555555555, 'Unit': 'Degree Celsius', 'month': 11}, {'Average': 12.06619623655914, 'Minimum': 10.551075268817204, 'Maximum': 13.581317204301076, 'Unit': 'Degree Celsius', 'month': 12}]]"]
Is the wildfire risk higher in Agros or in Lefke?	[["Data on wildfire-risk for Agros: {'Coordinates': [34.917436510667926, 33.01845097395608], 'Distance': 23.088961058374785, 'Risk': 'Very Low Risk', 'Score': 1, 'Unit': 'm'}", "Data on wildfire-risk for Lefke: {'Coordinates': [35.136626062654386, 32.85136385645825], 'Distance': 30.776583978546427, 'Risk': 'Very Low Risk', 'Score': 1, 'Unit': 'm'}"]
Is Nicosia or Limassol situated at a higher elevation?	[["Data on elevation for Nicosia: {'Coordinates': [35.1746503, 33.3638783], 'Elevation': 141, 'Unit': 'm'}", "Data on elevation for Limassol: {'Coordinates': [34.6852901, 33.0332657], 'Elevation': 21, 'Unit': 'm'}"]

Table C1: Containing questions and relevant data used for evaluating the end system and the answering LLM.