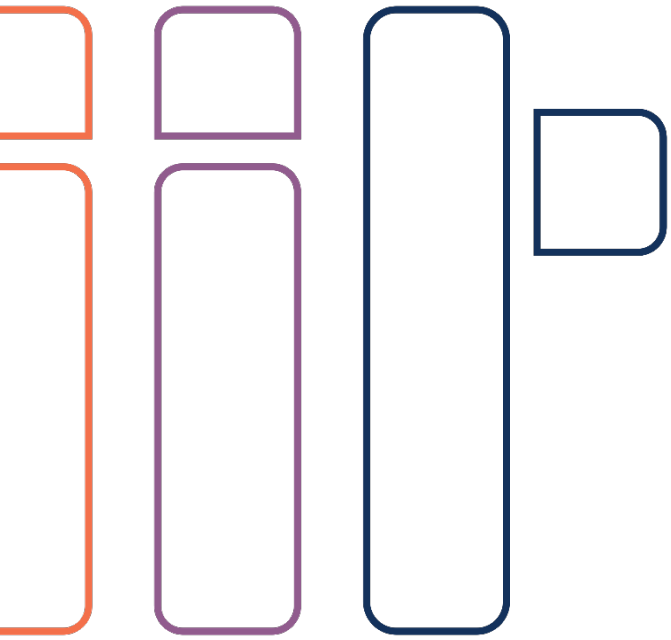

TECHNICAL REPORT

IIT TR-03/2025

MeteoChat: Semi-Automated Generation of Environmental Reports with LLMs and RAG

A. Scariot, A. Lo Duca, R. Lo Duca, A. Marchetti

MeteoChat: Semi-Automated Generation of Environmental Reports with LLMs and RAG

Alessandra Scariot (University of Pisa)
Angelica Lo Duca (IIT CNR)
Rosa Lo Duca (ARPA Lazio)
Andrea Marchetti (IIT CNR)

Abstract. Monitoring natural phenomena and supporting timely decision-making during emergencies or natural disasters is closely linked to a detailed analysis of the available environmental data collected over the years. Due to the large volume of data available, manually producing high-quality reports requires significant time and resources. This paper presents a system, named MeteoChat, designed to automate the creation of environmental reports by leveraging Large Language Models (LLMs), which are optimized through fine-tuning techniques and Retrieval-Augmented Generation (RAG). The system operates in two main phases: in the first phase, an environmental expert defines a set of key questions and corresponding answers applicable to various types of data, such as temperature and precipitation. This information serves as the foundation for fine-tuning the language model to specialize in the analysis and generation of environmental content. In the second phase, the optimized model is integrated into an RAG-based chatbot that combines specific data to generate accurate responses to be included in the reports. Users interact with the system through an intuitive web interface and can download the final report in docx format, containing all the requested information. This approach significantly reduces the time and resources needed for report generation while maintaining high-quality standards.

Keywords. Data Analysis, LLM, Report Building, Environmental Monitoring

ACM Classification. H.4 INFORMATION SYSTEMS APPLICATIONS

Table of contents

1. Introduction

The presence of large datasets related to the environmental domain requires the creation of high-quality reports that allow these data to be transformed into information and that are helpful for meteorologists in monitoring the environmental situation. The creation of these reports is complex and can be time-consuming. To help the experts build such reports, we have designed and implemented a system, called MeteoChat, based on artificial intelligence (AI) to generate semi-automatic reports. This system aims to create the skeleton of a report more quickly. Once generated, the experts can review and correct it. We implemented the system with the supervision of an expert in environmental monitoring.

This system is adaptable to different types of audiences, including general audiences, decision makers, and, most importantly, experts in the field. The questions differ in form and intent, but the system can respond accurately to different inquiries. The system is based on generative artificial intelligence, i.e., Large Language Models (LLMs), and is capable of processing environmental data, particularly those related to temperature, pressure, and precipitation.

The system implementation required several phases:

1. Study of the quality of the LLM responses: we conducted a study to test whether increasing the number of chunks, which are small portions of text, increased the quality of LLM responses, using Prompt Engineering techniques.
2. Data preparation: on the one hand, the raw data in the form of tables are queried through a list of general questions and answers related to environmental data. This list of questions was used to fine-tune the LLM model; on the other hand, we carried out an Extract, Transform, Load (ETL) process to analyze and divide data into multiple small datasets, each corresponding to a specific environmental metric.
3. LLM Fine-tuning: we used the OpenAI GPT-4o[1] model as the LLM model. We fine-tuned the model with meteorological data to specialize it in the environmental domain.
4. ChatBot integration: we integrated the fine-tuned model into a chatbot implemented using Flask[2], allowing end users to interact with it. In this phase we applied the two stages of RAG: the retrieval phase, where the system selects relevant data based on user query, and the generation phase, where the system applies the prompt template to the retrieved documents and uses the fine-tuned LLM to generate the final answer, which is then returned to the user.
5. Report Download: The final phase enables users to download the entire conversation and convert it into a report: the questions and answers are transformed into textual paragraphs, while AI automatically generates the title, abstract, and a summary table of the data.

The remainder of this work is organized as follows. In Section 2, we describe the background of this study, focusing on RAG, LangChain, Fine-tuning, and ChatBot implementation. Section 3 defines the problem setting, highlighting the current challenges for experts in generating weather reports manually and describing the environmental data used in our case study. Section 4 presents the architecture of the proposed system, describing each step in detail. Section 5 describes the code organization, detailing the structure and purpose of each file. Section 6 shows preliminary experiments of the proposed system. Lastly, Section 7 ends the paper and outlines future directions.

[1] OpenAI. *GPT-4o System Card*. OpenAI, 2024, https://openai.com/index/gpt-4o-system-card/
[2] Pallets. *Flask Documentation*. https://flask.palletsprojects.com/en/stable/

2. Background

This section provides a quick overview of how the basic technologies used in this study work. Specifically, we focus on RAG, LangChain, Fine-tuning, and ChatBot implementation. It is important to note that research in this area continuously evolves, as the technology behind these systems constantly advances.

2.1. RAG System

The RAG system is a technique used in Generative AI to increase reliability and trustworthiness of the outputs produced by LLMs[3], by retrieving relevant data from external sources at the time of user's requests.

Generic LLMs rely on statistics to generate responses, that is, they predict next words based on context, they do not verify the facts in their responses. For this reason, hallucinations represent one of the critical issues of these systems, i.e., responses that confidently present information that is actually invented or inaccurate. These systems, when they do not know the answer, lie with confidence. To overcome this problem, we used the RAG technique, which is one of the most widely used in the field of LLM. RAG provides the language model access to the non-parametric memory, a set of information that is not internal to the LLM but is retrieved through external retrieval mechanisms, such as a search engine or database.[4] In this way, the language model produces more grounded and factual answers and reduces hallucinations, because the LLM does not remain limited to its internal knowledge but retrieves information from external sources.

RAG's technique can separate irrelevant documents from relevant ones (noise robustness), does not give answers when there is no relevant information (negative rejection), and rejects inaccuracies. It assimilates information from multiple sources and can handle different types of queries.

Despite its various functionalities, RAG has deficiencies related to weaknesses in the language model: in fact, in some cases, it does not consider context, while in others, to answer a question, it collects information that is not relevant to the context.[5]

Figure 1 shows how a RAG system works. A RAG system includes two phases: retrieval and content generation. In the retrieval phase, relevant information is searched and then selected by the algorithms based on the prompt or user requests. This collection of external sources is added to the prompt and subsequently transferred as input to the LLM. In the generation phase, the language model uses the enriched prompt to develop a response appropriate to the context formulated by the user.[6]

Through a RAG system, the LLM responds more accurately, relying on both pre-existing knowledge of the model and information retrieved in real time.

[3] Kimothi A. *A Simple Guide to Retrieval Augmented Generation*. Shelter Island: Manning Publications, 2024.
[4] Kimothi A. *A Taxonomy of Retrieval Augmented Generation*, 2024.
[5] Kimothi A. *A Simple Guide to Retrieval Augmented Generation*.
[6] Martineau K. *What is retrieval-augmented generation?*. 2023,
https://research.ibm.com/blog/retrieval-augmented-generation-RAG. (Last Access 2025/04/24)

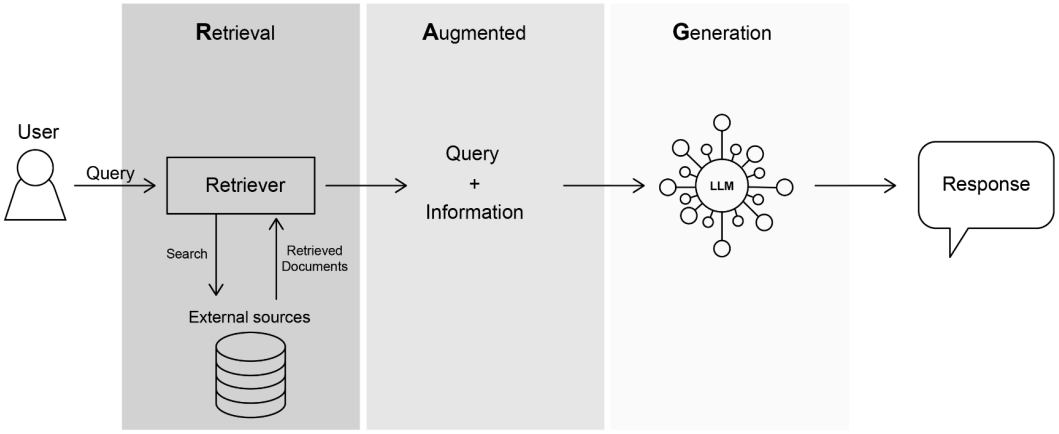

Figure 1. Example showing how the RAG system works.

2.2. LangChain

LangChain is an open-source framework for developing applications based on Large Language Models (LLMs). LangChain includes LangSmith, which is used for debugging, testing, and monitoring applications, and LangGraph, which specializes in the creation of static and multi-agent systems.[7] Additionally, LangChain is natively integrated with OpenAI models through the `langchain-openai` package. LangChain is primarily used to simplify the process of building, verifying, and deploying LLM-based applications. It can handle complex workflows, access data sources, and enable LLM applications to interact with external tools.[8]

Using LangChain provides different benefits, including the provision of a standard interface for key components, which facilitates switching between providers; orchestration, which allows different components to be connected into control flows that perform various actions; and observability and evaluation tools, which help developers monitor their applications.[9]

However, LangChain also has some drawbacks: as a continuously evolving library, new features are frequently added, sometimes requiring updates to existing functionality due to deprecation; furthermore, since it is tightly integrated with OpenAI, using it with alternative models, such as on-premise open-source models, can be challenging.[10]

[7] LangChain Documentation. *Introduction*, https://python.langchain.com/docs/introduction/ (Last Access 2025/04/27)
[8] Infante R. *AI Apps with LangChain*. Shelter Island: Manning Publications, 2024
[9] LangChain Documentation. *Why LangChain?*, https://python.langchain.com/docs/concepts/why_langchain/ (Last Access 2025/04/27)
[10] Mansurova M. *Topic Modelling in production*. 2023, https://towardsdatascience.com/topic-modelling-in-production-e3b3e99e4fca/ (Last Access 2025/04/27)

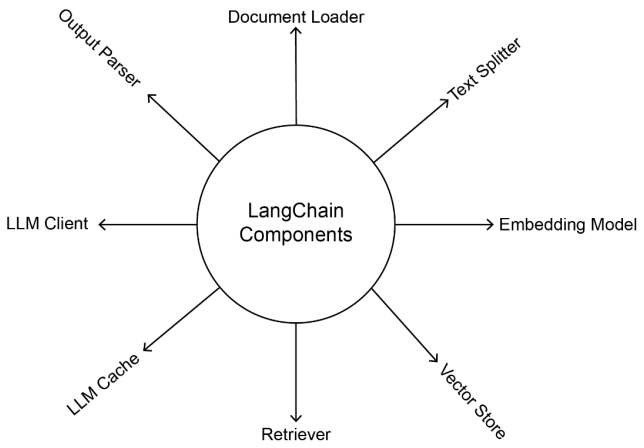

Figure 2. Example showing LangChain's Components.

The framework's architecture comprises several components, illustrated in Figure 2: external data is imported into a document using a Document Loader; the data is then divided into chunks by a Text Splitter. These chunks are subsequently vectorized using an Embedding Model and stored in a Vector Store. Chunks' retrieval is handled by a Retriever, which passes them to an LLM through a Prompt, while other Retrievers may obtain data from structured sources. An LLM Cache layer checks whether an LLM request has already been executed and, if so, returns the cached response. An Output Parser converts the LLM response into the preferred format.[11]

Vector databases are capable not only of storing text and its embeddings but also of supporting data updates. These databases enable similarity searches, returning chunks of text that are semantically closest to the user's query. Nevertheless, they still require a language model to process both the original query and the retrieved chunks. All these elements contribute to enhancing the scalability and performance of the system. In this study, Chroma[12], a vector database available in LangChain, was used to store and retrieve documents divided into chunks.

2.3. Fine-tuning

Fine-tuning is an additional training process applied to a pre-trained language model using a domain-specific dataset. This phase refines the model's capabilities, adapting it to specific needs through a smaller, curated dataset.[13] The dataset used for fine-tuning consists of input-output pairs, where each sample includes a prompt and the suggested completion.

In data reporting, fine-tuning can be used to customize the dataset to enhance the system's performance in analyzing, interpreting, and responding to questions concerning environmental data. The effectiveness of fine-tuning largely depends on the specificity and relevance of the dataset used: the more tailored the data is to the specific domain, the better the model's performance will be.[14]

[11] Infante R., *AI Apps with LangChain*, cit.
[12] *Chroma*, https://www.trychroma.com/.
[13] Infante R., *AI Apps with LangChain*, cit., p. 4
[14] Lo Duca A. *Data Storytelling with Altair and AI*. Shelter Island: Manning Publications, 2024, p. 195-201.

Fine-tuning achieves higher-quality results than prompt engineering alone: the model can be trained on more examples than can be included in a prompt. Prompts are processed with very fast response times, and fewer tokens are used due to shorter instructions.[15]

The fine-tuning process consists of seven main phases: a first phase of data preparation, in which data collection and cleaning are carried out; in the second phase, the model is initialized, choosing which pre-trained model to use (GPT-4o was used in the case under study); the third phase consists of organizing the training, while the fourth phase is the most important because it is the one in which the actual fine-tuning is performed: the model is trained on a specific domain based on the previously selected data. The next phase performs validation and evaluation of the model using specific metrics such as accuracy or F1-score. In the penultimate phase, the model is implemented by adapting it to concrete uses; for example, in this research, it is used to supply the domain-specific chatbot. The last phase consists of continuous monitoring of the model with periodic retraining steps to ensure reliability and relevance to the specific domain.[16]

2.4. Chatbot Implementation

A chatbot is an AI-powered software application that simulates human-like conversations, enabling automated user interactions through text-based responses. By leveraging natural language processing (NLP) and machine learning techniques, chatbots can interpret user queries, provide relevant information, and execute predefined tasks. In this study, we developed a web-based chatbot interface to facilitate interactions between the system and environmental experts. This enables them to refine queries, retrieve specific information, and generate reports in docx format.

There are various types of chatbots: FAQ bots that provide an answer directly to the user's question, process-oriented or transactional solutions in which the bot guides the user through a series of questions in order to achieve a goal, and routing agents that have to understand the user's initial intent and then they must decide whether to redirect the user to another specialised bot or to a human agent. [17]

A conversational AI usually follows three steps:

1. Understanding what the user wants: the user makes the request in natural language and a machine learning algorithm receives the message and determines the intent. The intent guides the next stage of the process.
2. Gathering the additional information needed to fulfil that desire: often the initial information provided by the user is not sufficient to complete the request and a dialogue engine comes to the aid of the user, asking clarification questions or interacting with other systems through API calls. The dialogue engine manages the state of the conversation and applies logic to respond to the user.
3. Giving the user what he wants: the conversation ends when the user receives the solution to his request.

These steps may vary depending on the type of chatbot: a question answering chatbot rarely uses the API while a routing agent does not fulfil the user's request but directs the user to the correct specialist.[18]

---

[15] OpenAI Documentation. *Fine-tuning*, https://platform.openai.com/docs/guides/fine-tuning (Last Access 2025/04/28)

[16] Noor F. *Seven Stage Fine-Tuning Pipeline for LLM*. 2024, https://medium.com/@noorfatimaafzalbutt/seven-stage-fine-tuning-pipeline-for-llm-ba16532cce65 (Last Access 2025/04/28)

[17] Freed. A. R., Jacobs C., Rozsa E. *Effective Conversational AI: Chatbots that work.* Shelter Island: Manning Publications, 2024, p. 1-2.

[18] Ibid., p. 4-6.

3. Problem Setting

Generating weather reports requires experts to spend a great deal of time and resources; an artificial intelligence-based system for semi-automatic weather report generation is a key innovation in climate data analysis and reporting. The system is capable of collecting, processing and interpreting so much data and transforming it into information. The analyzed data, collected between 2013 and 2023 by the micrometeorological network of ARPA Lazio, came in particular from station AL007 located in via Boncompagni 101, Rome; nine instruments were used for the measurements: an ultrasonic anemometer, a rain gauge, a thermo-hygrometer, two radiometers, and two pygeometers. The expert had to manually extract average, minimum and maximum of each metric's values (temperature, precipitation, pressure, …), as well as the mode for each month and year. Their objective is to build annual reports about metrics. This process is slow and, often, repetitive, thus we decided to develop a system to simplify the expert's work: using AI, the system can answer questions about weather data and generate detailed reports quickly and accurately.

4. System's Architecture

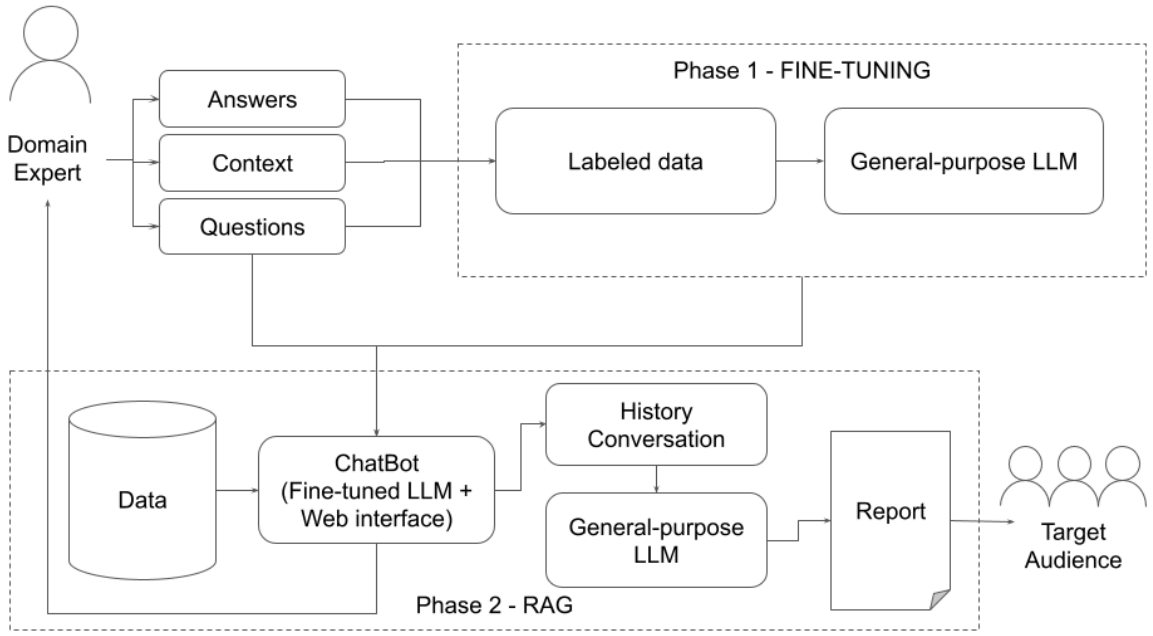

Figure 3. The proposed system involves two phases: the fine-tuning phase and the RAG phase. In the first phase, the LLM is refined based on the list of questions, context and answers that the domain expert defines. In the second step, the refined model answers the same questions related to specific data through a chatbot.

Figure 3 illustrates a two-phase system designed to create a domain-specific chatbot capable of answering targeted questions and generating tailored reports. The two main phases are:

1. *Phase 1: Fine-Tuning*: A domain expert begins by defining a set of generic questions, context and corresponding answers that reflect the types of queries a specific target audience might ask. These question-context-answer trios are based on the expert's understanding of the domain and the audience's informational needs. This data is used to create a labeled dataset, which is then used to fine-tune a large language model (LLM). The result is a refined version of the LLM that is optimized to handle audience-specific queries, ensuring that the model can generate accurate and relevant responses.

2. *Phase 2: RAG (Retrieval-Augmented Generation)*: The fine-tuned model is integrated into a RAG-based web chatbot system. This chatbot combines the refined LLM with a retrieval mechanism that allows it to access and incorporate information from a domain-specific dataset. When a user (e.g., the domain expert or end user) inputs a question, the system retrieves relevant data and generates a context-aware response using the LLM. The chatbot then produces a report containing the answers derived from the combination of the user's question, the expert's knowledge, and the relevant dataset.

A major benefit of this setup is that the model does not need to be retrained when switching datasets, as long as the same question structure is used. This enables the system to be flexibly reused across various projects, thereby improving efficiency and scalability.

4.1. Analyzing Outputs Based on Chunk Size Variations

During the RAG phase, each text is divided into chunks, where a chunk is a predefined portion of text, typically chosen based on parameters such as the number of words, characters, or semantic units. The chunking process is crucial for effectively building the knowledge base of RAG systems and is expected to have a significant impact on their quality. According to a study, a large chunk provides better context but increases the amount of noise, while a small chunk offers more precise information but often overlooks important details.[19]

For this reason, we conducted a preliminary study to verify whether the model's response changes as the chunk size varies.We analyzed GPT-4 and GPT-4o.

To determine the optimal chunk size for our system, we varied it from 500 to 10,000 (with increments of 100 up to 3000 and 500 beyond 3000). We then evaluated the performance of the considered models.

We asked five questions:

1. "In which month and year was the highest temperature value recorded?": the model's answers were set equivalently except in two cases where the model gave a shorter answer (chunk_size=2200 and chunk_size=4500). Regarding correctness, most of the answers were correct (27 out of 40) while the remaining ones gave a high temperature value but not the highest among the available data. In this case, it seems that chunk variation did not affect the quality of the answer.
2. "What was the hottest month for each year?": starting with chunk_size=5500 more precise answers were given (the hottest month of four years is given while in the other cases only three). It appears that the variation in chunk size was more influential in this question.
3. "You are tasked with analyzing temperature data to determine the hottest month for each year. For example, if we have three months-May with an average temperature of 24°C, June with an average temperature of 25°C, and July with an average temperature of 26°C-the hottest month is the one with the highest average temperature. Based on the given data, what was the hottest month for each year?": this is a refinement of the previous question using a Prompt Engineering technique called Prompt One-Shot, in which the system is given an example of the task already performed. This should increase the quality of the answer, and in fact in most cases we get a more detailed and accurate answer. When varying the chunk size, the GPT-4 model produced answers that remained basically unchanged; the only differences involved the approximation of numerical values (for example sometimes it returned 26 °C and other times 26.511698423577794 °C). In contrast, the GPT-4o model showed more significant variations: in some cases, it generated short and concise responses, while in others, it provided more precise answers with all the reasoning steps included. However, the chunk size does not influence these differences.
4. "What is the percentage increase in temperature over the decade 2013-2023?": in this instance, the GPT-4 model provided values that were completely different from the actual percentage increase and generated rather long answers (exceptions occurred with chunk_size= 700, chunk_size= 800 and chunk_size= 10000 where the model produced significantly shorter

[19] Kimothi A. *A Taxonomy of Retrieval Augmented Generation*.

answers). GPT-4o returned slightly better answers and did not change its behavior with variations in chunk size. In each case, it provided a detailed explanation of the reasoning behind the answer, even though it returned incorrect and very discordant values.

5. "The formula for the percentage increase is: [(Average temperature of year B - Average temperature of year A)/Average temperature of year A]*100. What is the percentage increase in temperature over the decade from 2013 to 2023 (in this case year A is 2013 and year B is 2023)? If any months are missing from the provided data, exclude them from the calculation. Let's think step by step.": in this question, we used the Prompt Engineering technique called Chain of Thought to carry out step-by-step reasoning to obtain more accurate and detailed answers. The GPT-4 model provided responses accompanied by explanations of the reasoning and varied greatly in length regardless of the chunk size. However, the model still gave incorrect percentage increase values that differed significantly from each other. In some cases (chunk sizes of 2100, 2400, 3500, 5000, and 7000), the model did not perform the calculation but instead asked the user to do it. In contrast, GPT-4o produced significantly longer and more accurate answers. In five responses, the model returned the same value, that is a percentage increase of 4.30% (in responses with chunk sizes of 500, 2000, 6000, 8000, and 8500), while in three other answers (chunk_size = 4000, chunk_size = 5500, and chunk_size = 7500), it indicated a value of 4.77%. This greater agreement in the results suggests that applying the Chain of Thought technique improved the quality of the responses for the GPT-4o model.

Table 1 summarizes how chunk size affects the quality of responses in the two models analyzed, GPT-4 and GPT-4o.

	Question	**Technique**	**GPT-4 Impact on Chunk Size**	**GPT-4o Impact on Chunk Size**	**Overall Evaluation**
1	Month and year with maximum temperature	None	No major impact. Responses mostly correct (27/40)	No major impact	Solid to chunk size
2	Hottest month for each year	None	Responses improve with chunk_size above 5500	Responses improve with chunk_size above 5500	Influential chunk size
3	Previous question, but with example	Prompt One-Shot	Answers stable, only numerical accuracy varies	Significant variations, but not dependent on chunk size	Prompt effective, chunk size irrelevant
4	Percentage increase in temperature (without formula)	None	Answers incorrect and long, exceptions with very low or maximum chunk_size that were short	Responses improved but still incorrect, little impact from chunk size	Incorrect responses from both models
5	Percentage increase in temperature with formula	Chain of Thought	Longer answers with discordant values. Often does not do calculation, but requires the user to perform it.	Detailed and accurate answers, greater agreement in results	Prompt very efficient on GPT-4o

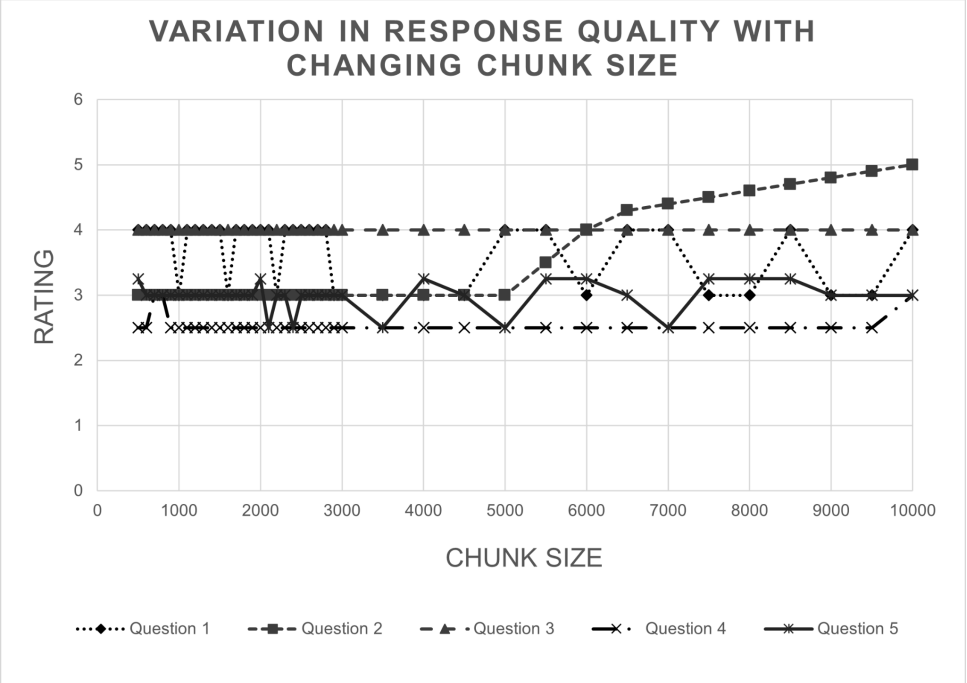

Figure 4. Graph illustrating how the quality of answers (evaluated on a rating scale of 1 to 5) varies in relation to chunk size, with separate lines for different questions.

The graph illustrates the impact of varying chunk sizes on response quality for each question. For each model, we ranked the response quality of each question based on a 5-point Likert scale (5 = correct answer, 1 = totally wrong answer). We manually assigned the scores based on the accuracy and completeness of the answers provided by the system. To create the graph, we calculated the average score for each query by combining the results of both models, GPT-4 and GPT-4o. This approach provides a summary representation of the overall average quality of responses for each question as the chunk size varies.

As can be observed only for some types of questions, the variation in chunk size seems to influence the quality of the answers, although the differences are not particularly significant.

Therefore, for implementing the chatbot, we chose a chunk size of 1000 because it represents a balanced intermediate value that provides a reasonable trade-off between preserving enough contextual information and maintaining efficient processing.

4.2. Fine-Tuning

There are two strategies to prepare the dataset for fine-tuning: manual building and building from sources. Manual construction is based on the manual definition of each pair (prompt, completion). It allows for obtaining more accurate results, as it specifies exactly how the model should behave given a specific input. The main disadvantage is the high time cost required to manually write each pair of data. First, a set of examples is defined, at least 10 for OpenAI models, and the model is trained on these. Then, the performance of the trained model is evaluated by comparing it with the original one. If the results produced by the new model are worse than or equal to the original model, new examples are added or existing ones are improved, and the procedure is repeated until a fine-tuned model that outperforms the original one is obtained. [20]

The strategy of building from sources involves creating a dataset using external sources written in a language specific to a particular domain. When using this technique, it is essential to check the data license.[21] This approach enables users to tailor the model to specific fields, ensures accuracy by utilizing reliable and precise information from external sources, and contributes to enhancing overall performance.

[20] Lo Duca A. *Data Storytelling with Altair and AI*. Shelter Island: Manning Publications, 2024, p. 195-201.
[21] Lo Duca A. *Data Storytelling with Altair and AI*. Shelter Island: Manning Publications, 2024, p. 201-203.

For this research, we adopted the manual approach.

The preparation of the model for fine-tuning involved, on the one hand, a set of questions, their context and answers that needed to be provided to the model to train it for a specific task; and, on the other hand, processing the raw data using the Extract, Transform, Load (ETL) process.

4.2.1 Q&A for the Model

To create the fine-tuned model, we defined a set of generic questions, their context and the expected responses that are customized according to the data.

Table 2 shows the questions, along with their context and completions, on which the model was fine-tuned.

Question	Context	Answer
In which month of the year Y was the highest value of the examined metric recorded for station S?	Find the maximum value in the measurement column and return the corresponding month(s).	The highest value was recorded on month X of year Y.
By how much do the maximum and minimum values of the examined metric differ in year Y for station S?	Find the maximum and minimum values and calculate the difference.	The maximum and minimum values differ by X units.
How many times did the metric drop below value X at station S in year Y?	Count how many measurements are below 0°C.	The temperature dropped below 0°C X times in year Y.
What is the average annual value of the examined metric in year Y for station S?	Calculate the average of all measured values.	The average annual value of the parameter in year Y is X.
What was the most frequent value (mode) of the examined metric in year Y for station S?	Find the most frequently occurring value.	The most frequent value of the parameter in year Y was X.
In year Y, how many measurements of the examined metric were not recorded due to technical issues with station S?	Calculate missing measurements assuming 48 per day for 366 days in a leap year.	The number of measurements not recorded due to technical issues was X in year Y.
In which month of the year Y was the lowest value of the examined metric recorded for station S?	Find the minimum value and return the corresponding month(s) that correspond to it.	The lowest value was recorded on month X of year Y.
By how much has the average value of the examined metric changed over the last two years (Y1 and Y2) for station S?	Calculate the average for each year, then compute the difference.	The average value changed by X units between 2023 and 2024.
When ordering the dataset of the examined metric in ascending order, what is the median value for year Y for station S?	Sort the values and calculate the median.	The median value of the parameter in year Y is X.
In which month of year Y does the examined metric show the greatest discrepancy between its maximum and minimum values for station S?	For each month, calculate the difference between the maximum and minimum values. Return the month with the highest difference.	The month with the greatest discrepancy is X.

These ten questions, the minimum required by OpenAI models, including their context and expected answers, were formatted into a JSONL file. This file contains a list of messages, each with a specific role and content. There are three possible roles: system, which sets the context or behavior of the assistant; user, representing messages sent by the user; and assistant, which represents the assistant's (i.e., model) answers. An example of the messages is illustrated in the Implementation section.

The JSONL file is given as input to the model for fine-tuning it on a specific domain, namely the environmental domain in this study.

At the outset of the research, we outlined a list of preliminary questions that served as the basis for fine-tuning the model. Subsequently, an expert defined the final set of questions, shown in Table 2, which was used to fine-tune the model again with more specific and appropriate questions.

4.2.2 Implementation of the Fine-Tuned Model

To improve the model's performance for the specific task, we fine-tuned it using the Azure OpenAI service and selected GPT-4o as the base model.

We started the fine-tuning process by uploading a JSONL-formatted dataset containing a set of questions, procedures, and answers representative of the target domain. After the upload, we validated the dataset to ensure its accuracy and suitability for training. Once validated, we conducted the actual fine-tuning phase, which enabled us to adapt the model to the specific environmental domain.

As a result, we obtained an optimized model capable of generating more relevant and coherent responses, which we integrated into a chatbot.

4.3 RAG

The second phase of the system is based on the Retrieval-Augmented Generation (RAG) technique, which combines a fine-tuned model with a data retrieval mechanism to enhance response reliability and mitigate LLM hallucinations. This phase includes data preparation (ETL), user interaction through the chatbot, and automatic report generation.

4.3.1 Using ETL for Input Data

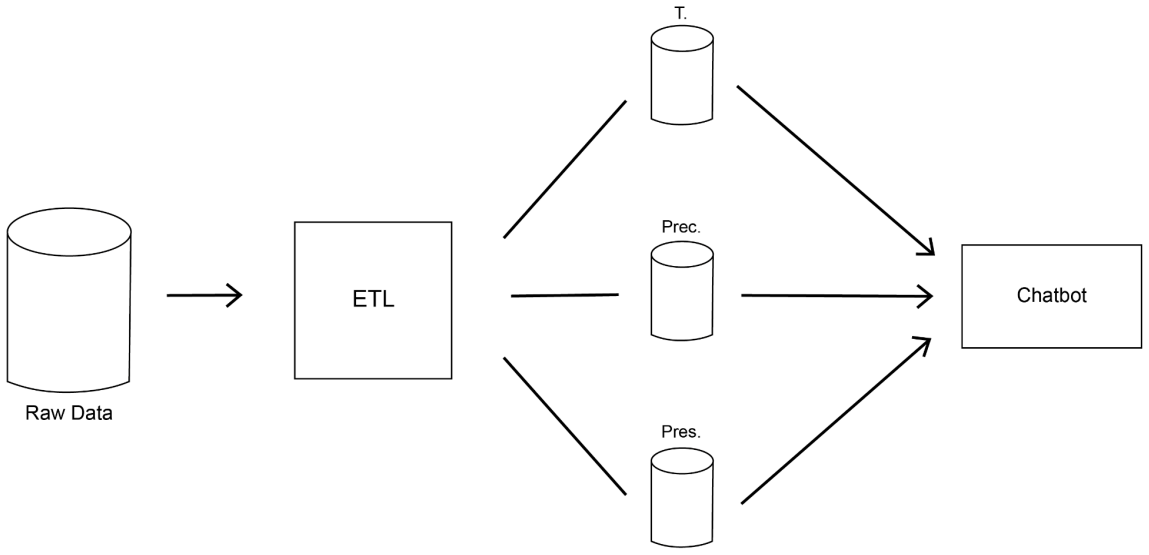

Figure 5. Diagram of the ETL (Extract, Transform, Load) process for data integration into a chatbot. Raw data is processed through the ETL system, which categorizes it into three storage types: "T" (Temperature), "Prec." (Precipitation) and "Pres." (Pressure). These processed datasets are then fed into the chatbot for user interaction.

Initially, the chatbot employed only temperature-related data. Then, we also included precipitation and pressure. To inject raw data into our RAG system, we introduced an Extract, Transform, Load (ETL) process. This process involves collecting and cleaning raw data and organizing it into smaller datasets, each corresponding to a specific metric (Figure 5).

For each year, ARPA Lazio created a dataset containing information on station code, date and time of measurement, temperature, relative humidity, wind speed, wind direction, precipitation, atmospheric pressure, global radiation, albedo, atmospheric infrared, terrestrial infrared, and net radiation.

Table 3 presents an extract of the original 2013 dataset before the ETL process. Please note that the value -9999.900 denotes the absence of data.

cod_staz	yyyymmdd_hhii	T	Ur	VV	DV	Prec	Pres	Rg	Rg_Down	L_Up	L_Down	Rn
AL001	20130101_0030	3.299	67.284	0.859	135.476	-9999.900	1020.84	-1.451	1.190	260.510	311.727	-53.858
AL001	20130101_0060	3.197	69.311	0.372	267.316	-9999.900	1020.65	-2.546	0.454	259.862	310.288	-53.427
AL001	20130101_0130	2.166	66.969	0.514	79.600	-9999.900	1020.61	-1.311	0.943	261.740	309.139	-49.654
AL001	20130101_0160	3.138	75.538	1.436	92.540	-9999.900	1020.43	-0.692	1.159	266.050	311.389	-47.189
AL001	20130101_0230	4.808	82.960	1.703	109.955	-9999.900	1020.13	-1.320	0.795	271.424	315.671	-46.362
AL001	20130101_0260	4.530	79.211	0.775	91.107	-9999.900	1019.99	-1.864	0.593	265.467	313.320	-50.310
AL001	20130101_0330	3.882	76.397	0.225	282.634	-9999.900	1019.78	-2.600	0.382	265.904	312.836	-49.915
AL001	20130101_0360	1.955	81.451	0.866	263.582	-9999.900	1019.72	-3.283	0.040	258.898	307.915	-52.340
AL001	20130101_0430	1.343	82.932	0.606	124.837	-9999.900	1019.53	-1.549	0.683	262.017	306.888	-47.103

We performed a data cleaning process for each metric, extracting the relevant data for every year. Table 4 shows an extract of the 2013 temperature dataset, after the data cleaning process.

date	T
2013-01-01 00:30:00	4.302
2013-01-01 00:59:00	3.967
2013-01-01 01:30:00	3.86
2013-01-01 01:59:00	3.595
2013-01-01 02:30:00	3.304
2013-01-01 02:59:00	3.011
2013-01-01 03:30:00	2.918
2013-01-01 03:59:00	2.876

2013-01-01 04:30:00	2.767

Through this approach, we created a separate dataset for each metric and each year. Subsequently, we further refined these datasets by computing statistical measures, including mean, median, mode, minimum, and maximum values. We performed these calculations using the Pandas[22] and SciPy[23] libraries.

Table 5 presents an extract of the temperature dataset for 2013, as detected by station AL007, following statistical calculations. Note that during October 2013 station AL007 did not detect any temperature data.

Year	Month	Mean	Max	Min	Mode
2013	1	8.648333080424887	17.383	1.057	4.52
2013	2	8.239770833333333	17.291	-0.411	3.033
2013	3	12.108737804878048	24.841	1.7	11.251
2013	4	17.177874357090374	30.288	7.941	11.957
2013	5	17.906406586021504	27.452	9.371	16.936
2013	6	22.177445855115757	34.246	12.839	14.35
2013	7	25.702890456989245	35.594	16.746	25.86
2013	8	26.511698423577794	37.312	18.864	21.737
2013	9	22.024836111111111	31.644	14.72	17.876
2013	10	No data Available	No data Available	No data Available	No data Available
2013	11	13.053358956276446	22.958	0.761	7.883
2013	12	9.441964381720432	19.413	1.151	5.428

The outputs are now ready to be provided as input to the chatbot. However, to optimize the speed of interaction, we selected only datasets related to temperature, pressure, and precipitation. As a result, the conversational agent can answer specific questions regarding these metrics.

[22] *Pandas*, https://pandas.pydata.org/.
[23] *Scipy*, https://scipy.org/.

4.3.2 The Chatbot

For the development of the chatbot, we used Flask[24], a lightweight and flexible web microframework for Python. We chose Flask because it enables the development of web applications simply and efficiently, providing essential functionalities such as HTTP request handling and routing without imposing a rigid structure.

The chatbot implementation uses an intuitive web interface that allows users to interact easily with the system (Figure 6). The home page features a simple and functional layout: at the center, the conversation area displays the messages exchanged between the user and the chatbot (A). At the bottom, a text input field (B) allows users to type their messages, accompanied by a send button (C) and a second button (D) that enables users to download a detailed conversation report.

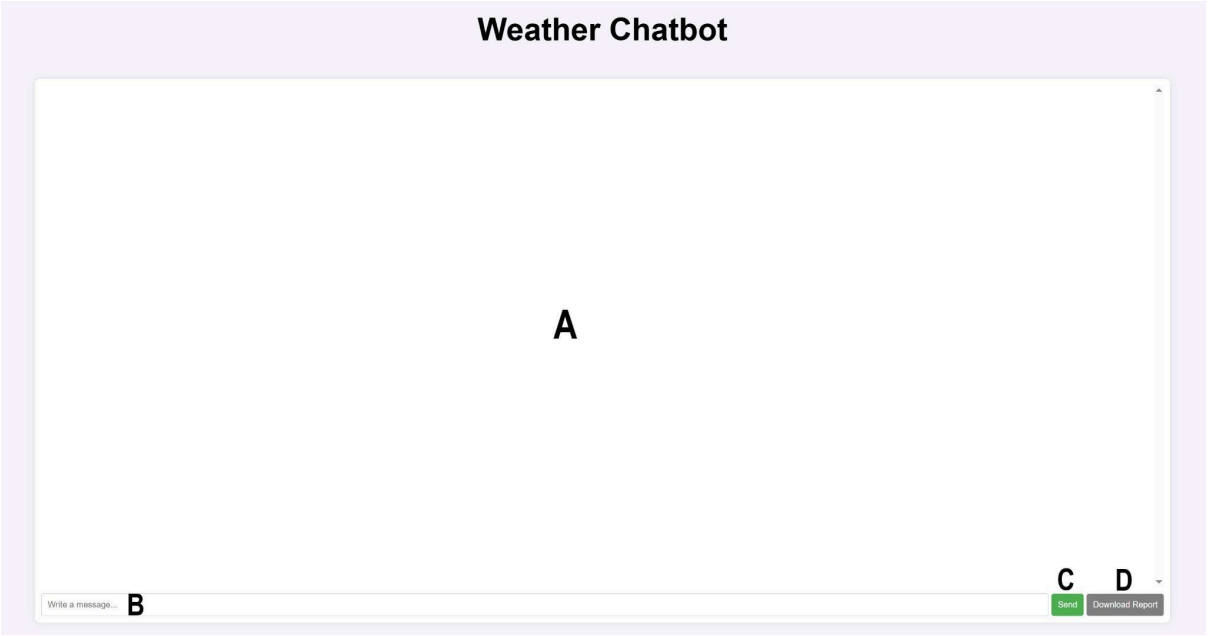

Figure 6. Chatbot's Interface: in the center is the conversation window (A), where the user can see the sent and received messages. At the bottom, there is a text field that allows the user to type in their questions (B), as well as the "Send" (C) and "Download Report" (D) buttons.

When the user writes and sends a message, the system processes it and generates a real-time response using the fine-tuned model. The chatbot understands the content of the message and provides dynamic answers based on the user's input. The system saves each message in the conversation history, enabling users to track their interactions.

It is important to note that the chatbot can respond to questions related to the environmental data it was trained on. If asked about unrelated topics, it may not be able to provide an appropriate response.

Another key feature is the ability to download a detailed report of the conversation. In addition to the exchanged messages, the document includes supplementary information, such as the title and abstract of the report. This aspect will be discussed in more detail in the following subsection.

The system runs on a local network through the Flask web server, which, thanks to its modular architecture and efficient request management, makes the application easily scalable and adaptable to different use scenarios.

[24] Pallets. *Flask Documentation*. https://flask.palletsprojects.com/en/stable/

4.3.3 Report Building

When the user has finished their queries and the chatbot has answered their questions, they can click on the “Download Report” button, which allows them to generate and download the full report of the conversation. First, the system verifies that a conversation is available; if not, an error message is shown. If the conversation is available, the system proceeds to create the document using the Python-docx library. [25]

Figure 7 shows the flow used to generate the report. Initially, the list of messages exchanged between the user and the chatbot is retrieved. Next, a general-purpose LLM is used to automatically generate a title and abstract that are coherent with the content of the conversation. The abstract is formulated to simulate the language used by a domain expert, making the document more professional and informative.

Next, each question is entered as a header, and its answer is entered as a paragraph. Finally, the LLM produces a summary table with the keyword representing the question in the first column and the numerical value contained in the answer in the second column. This table allows the relevant data from the conversation to be quickly highlighted.

At the end of the process, the system saves the document in the Microsoft Word format and makes it available for download. The process of generating the report may take several minutes, depending on the length and complexity of the conversation.

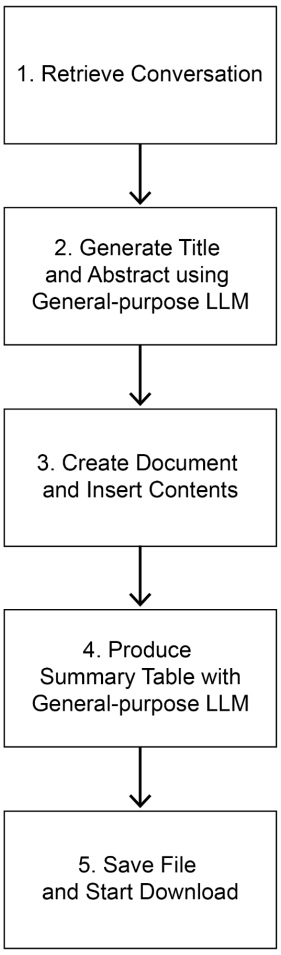

Figure 7. Report generation flow: the system retrieves the conversation, generates title and abstract using AI, creates the document with the contents, adds a summary table generated by AI and exports the file for download.

[25] *Python-docx*, https://python-docx.readthedocs.io/en/latest/.

5. Implementation

We have implemented the system using the Python programming language[26]. The system presents a structure divided into the following main folders and files that perform the different tasks:

- `data/`: folder that contains the data cleaned by data cleaning, which will later be used for statistical analysis.
- `output/`: a folder containing the datasets resulting from the application of statistical analysis, used as input for the chatbot.
- `source/`: folder that includes the original datasets, which are later extracted and cleaned.
- `chatbot.py`: script containing the code for implementing the chatbot.
- `data_cleaning.py`: code for data cleaning of temperature data.
- `data_cleaning_prec.py`: code for data cleaning of precipitation data.
- `data_cleaning_pres.py`: code for data cleaning of atmospheric pressure data.
- `Example Report.docx`: file containing an example of the final report generated by the AI.
- `fine-tune-model-azure_expert.py`: script for fine-tuning the model.
- `general-public_expert.jsonl`: file with the set of questions and answers used as input for fine-tuning.
- `index.html`: web interface of the chatbot.
- `secrets_sample.json`: sample file that contains the credentials required to access external services (in our case Azure).
- `statistics_prec.py`: code for manual statistical analysis of precipitation data.
- `statistics_pres.py`: code for manual statistical analysis of pressure.
- `statistics_temp.py`: code for manual statistical analysis of temperature.

Figure 8 shows a screenshot of the file system structure of the project, illustrating the organization of the folders and files described:

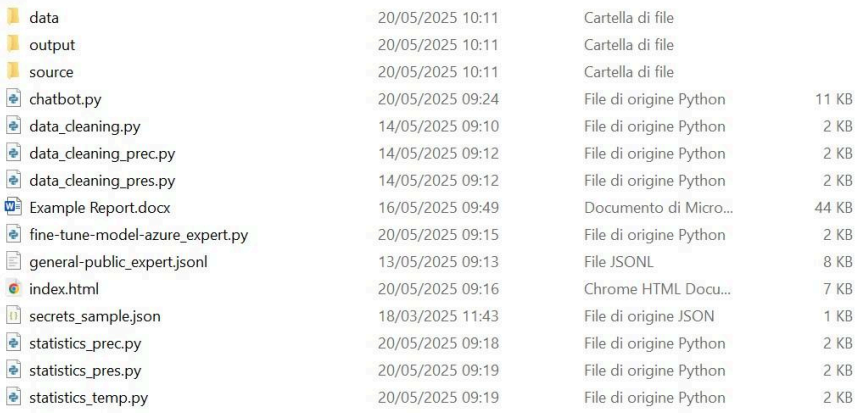

Figure 8. Structure of the project file system, with major folders and files.

5.1. chatbot.py

`chatbot.py` is one of the core scripts of the system because it implements the chatbot's functionality. It generates answers to user questions about the climate and creates the report in Microsoft Word format. Initially, after importing the necessary libraries and modules, the system loads environmental data (specifically precipitation, temperature, and pressure data) on which it had previously performed statistical calculations. The script then combines all documents and segments them into 1000-character chunks to simplify management.

[26] https://github.com/djinfouma/meteochat

It loads OpenAI ChatGPT[27] credentials to create embeddings, which convert documents into numerical representations that capture their semantic meaning. This step prepares the data for the retrieval phase of the RAG system.

If it doesn't exist, the script creates the indexed database and saves it locally using Chroma.

It defines a prompt template that provides the LLM the necessary context to answer environmental questions, including information about measurement units or response formatting. Then, it configures the language model by specifying the ID of the fine-tuned model, ensuring more accurate and context-aware responses.

The system is made available through the web microframework Flask[28], which offers two endpoints: one to handle the conversation and one to download the report. The last creates a Microsoft Word document that includes the title, abstract, questions with corresponding answers, and a summary table of the data. During this phase, the RAG generation occurs: the language model produces more detailed answers enriched with previously retrieved information.

5.2. data_cleaning.py

The `data_cleaning.py` script performs the cleaning of the original raw data contained in CSV files, one for each year from 2013 to 2024. This code uses the Pandas library to read the files from the `source` folder.

Subsequently, it selects the values for weather station AL007, extracting only the columns of station code, date/time, and the specific metric. This data cleaning process is performed for each metric, such as temperature.

The script removes missing or outlier values and formats the date correctly. Finally, it saves the cleaned files, one for each year, in the `data` folder.

5.3. fine-tune-model-azure_expert.py

This code, together with `chatbot.py`, constitutes one of the most essential parts of our system. It is responsible for transforming the generic LLM into a specialized model on environmental data.

After importing the required libraries and modules, the script proceeds to load the Azure OpenAI [29] credentials and then loads the file used to train the model (`general-public_expert.jsonl`).

After the file has been successfully processed, the fine-tuning phase takes place in which the model is fine-tuned on the generic LLM and the dataset we provided to it earlier. If fine-tuning ends successfully, the script returns the new model's id, otherwise, it returns an error.

5.4. general-public_expert.jsonl

We used this file as input for training the model, which contains a list of messages with three distinct roles defined (system, user, and assistant), along with their corresponding contexts.

Specifically, the system role explains the procedure the model must follow to generate a consistent response to the request; the user role specifies a possible user's question addressed to the conversational agent, while the assistant role provides the plausible answer to the user's query.

The following example shows the structure of the JSONL file:

```
{"messages": [
      {
            "role": "system", "content": "You are a data analyst showing
            data to the general public. Choose a parameter (e.g.,
            temperature, pressure, relative humidity…). Consider a table
            consisting of two columns: month and measured values of the
```

[27] OpenAI. *ChatGPT,* https://openai.com/chatgpt/overview/.
[28] Pallets. *Flask Documentation*. https://flask.palletsprojects.com/en/stable/.
[29] *Azure OpenAI*, https://azure.microsoft.com/en-us/products/ai-services/openai-service.

```
            chosen parameter. In the column of measured values, find the
            highest value. Once found, read the corresponding month in the
            other column. Return this month as the output.If the highest
            value appears in multiple cells, return all the corresponding
            months as the output.”
      },
      {
            "role": "user", "content": "On which month of the year Y was the
            highest value of the examined parameter recorded for station
            AL007?”
      },
      {
            "role": "assistant", "content": "The highest value was recorded
            on month X of year Y."
      }
]}
```

5.5. index.html

The interaction between the user and the chatbot occurs through this file, which serves as the web interface of the system.

The page is structured as a typical conversational interface, where the chat is displayed in real-time. The user can write and send new messages and download the conversation report via the download button.

The file consists of several sections: the header, which defines the page metadata and title, the body, which contains the main elements of the user interface, a section dedicated to JavaScript functions that handle the client-side logic, and finally, a part that defines the graphical appearance of the page.

5.6. secrets_sample.json

The `secrets_sample.json` file represents a JSON structure used to configure access to the different versions of the GPT LLMs via the Azure OpenAI service credentials.

Each section consists of three parameters for making calls to the models: API key, endpoint, and deployment name. The API key represents the authentication key needed to access the API service, the endpoint represents the URL of the service from which the API request is made, while the deployment name is the name used to identify the model.

5.7. statistics.py

This code processes the CSV files, which have previously been subjected to data cleaning, and performs statistical analysis on them. For each file, one for every year, the script reads the data through the Pandas library and groups them by month. The analysis is applied to each metric, such as temperature or pressure.

We performed calculations of descriptive statistics for each month: mean, maximum, minimum, and mode (calculated using the SciPy module). The result is a new CSV file containing the meteorological station code, year, month, and monthly statistics. Lastly, we used the set of files obtained through this process as input for the chatbot.

6. Preliminary Experiments

We conducted an experiment to evaluate the quality of the answers generated by MeteoChat. We selected three questions, formulated by the expert, for the experiment (Figure 9 shows two of the three questions submitted to the chatbot):

- Q1: Which month of the year 2020 had the highest temperature value recorded for station AL007?
- Q2: What is the average annual value of precipitation in 2014 for station AL007?
- Q3: In which month of the year 2023 does the pressure show the most significant discrepancy between its maximum and minimum values for station AL007?

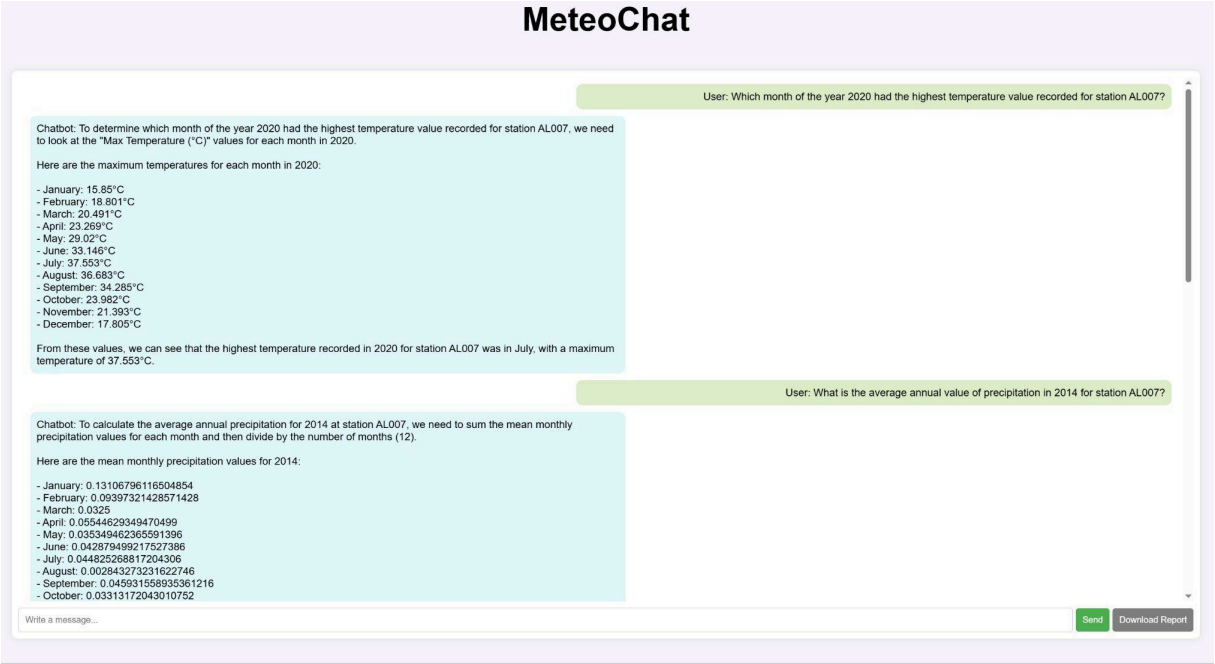

Figure 9. Example of a conversation in which the user asks the chatbot two questions, highlighted in green: one concerns which month of the year 2020 recorded the highest temperature for station AL007, while the other one relates to the average annual precipitation value in 2014 for station AL007.

We repeated the test five times, using five different conversations. On average, MeteoChat responded to the first question within 5 seconds. The second question had an average response time of about 28 seconds, while the third question took 6 seconds.

After submitting all three queries, we proceeded to download the report in Microsoft Word format by clicking the grey button located in the bottom-right corner (Figure 10). This operation took an average of 2 minutes and 38 seconds.

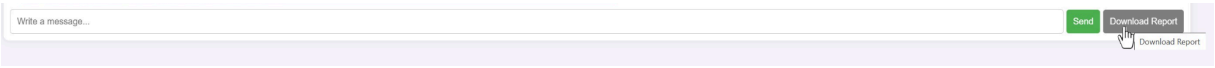

Figure 10. User interface following the activation of the Download Report function. Report generation may require several minutes.

Once the required time had passed, the report was automatically saved in the Downloads folder and could be accessed from the browser's download history (Figure 11).

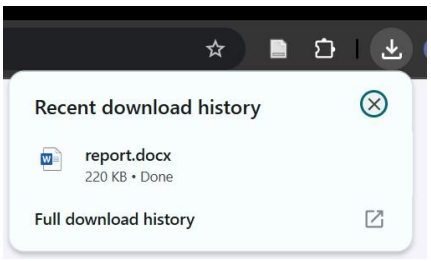

Figure 11. The Report file is downloaded via the browser and is visible in the history of recent downloads. The file is automatically saved in the default downloads folder of the operating system used.

The generated report follows this structure: the first page presents the title and the abstract generated by the AI. The following pages include the questions asked by the user during the conversation, along with the corresponding answers provided by the chatbot (Figure 12).

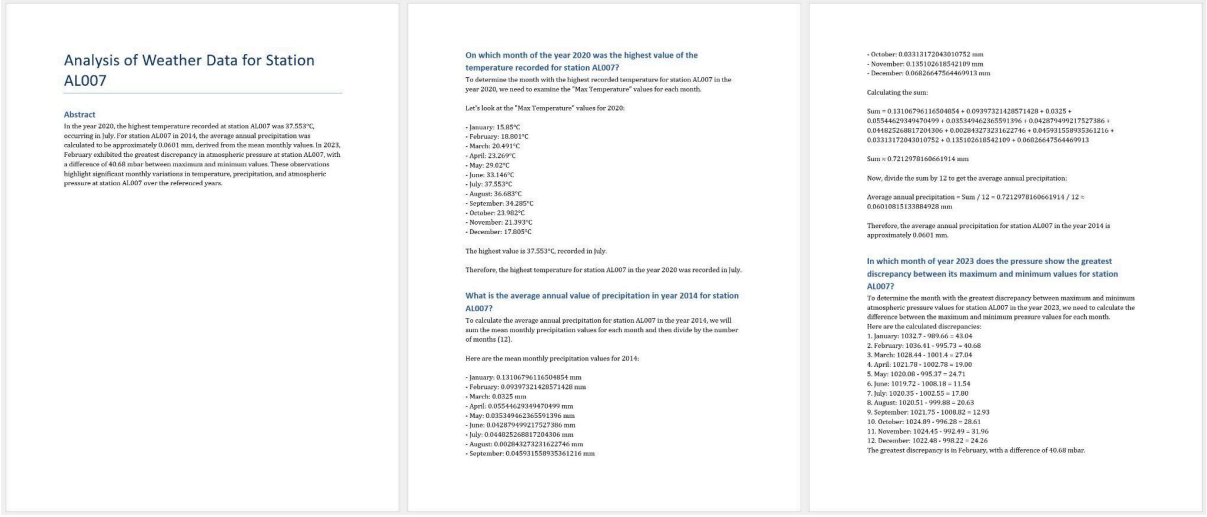

Figure 12. Example of the Final Report: the left page displays the title and abstract generated by AI, while in the middle and right pages show the set of questions and answers of the conversation.

The last page, shown in Figure 13, contains a summary table: the left column displays a keyword summarizing each question. In contrast, the right column shows the main information extracted from the corresponding response. This table provides a quick overview of the conversation content, enabling users to easily identify key topics and relevant answers.

Summary Table

Question	Numerical Data
Keyword	Value
Highest Temperature Month	July
Average Annual Precipitation	0.0601
Greatest Discrepancy Month	February

Figure 13. Example of the summary table of the Final Report: the first column contains keywords summarizing the questions, while the second column displays the corresponding numerical values or key information extracted from the answers.

To run the experiments, we used a computer equipped with an Intel Core i5-6400 processor running at 2.70 GHz and based on x64 architecture. The device includes 16 GB of RAM and three storage units: a 224 GB SSD, supported by two HDDs, each with a capacity of 932 GB and 466 GB,

respectively. The graphics card installed is the NVIDIA GeForce GTX 1660 SUPER with 6 GB of dedicated video memory. The operating system is Windows 10 Pro, specifically the 2009 version, 64-bit.

7. Conclusion

The realization of MeteoChat, based on the combination of AI, RAG, and fine-tuning, achieved the dual objective set at the beginning of the research: on one hand, speeding up the work of the environmental technician, and on the other, generating accurate and high-quality reports.

Through the study of the quality of LLM responses, the analysis revealed that using an intermediate value for chunk size is the optimal choice to maintain contextual information and process efficiency. At the same time, the preparation of the data, through ETL and data cleaning procedures, enabled the selection of the most relevant information, which helped create consistent and optimized datasets for the subsequent fine-tuning phase.

Fine-tuning produced a model specialized in environmental data that can provide more coherent and accurate answers compared to a general-purpose LLM.

The system was also enhanced through the use of the RAG method, which improves the reliability of the answers and reduces the hallucination, a phenomenon typical of generic language models.

The final phase of report creation simplifies the meteorologist's work by semi-automatically generating the report based on the conversation with the chatbot. It's essential to emphasize that this system is not intended to replace expert work, but rather serves as a powerful tool to expedite the report creation process.

This first attempt to combine AI, RAG, and fine-tuning produced satisfying results, opening the way for new potential developments: extending system evaluation to other domain experts, enhancing the report by adding charts, and increasing textual content.

Finally, a further development will enable MeteoChat to operate fully automatically through the use of multi-agent technology, i.e., systems consisting of multiple agents that collaborate to design and solve complex and dynamic tasks autonomously, thereby reducing human intervention and simplifying the meteorologist's work.

References

1. ARPA Lazio, *Dati rete micro-meteorologica*, 2013-2023. Rete micro-meteorologica - ARPA Lazio. https://www.arpalazio.it/rete-micro-meteorologica (Last Access 2025/03/19).
2. *Azure OpenAI*, https://azure.microsoft.com/en-us/products/ai-services/openai-service (Last Access 2025/06/13).
3. *Chroma*, https://www.trychroma.com/ (Last Access 2025/05/20).
4. Freed. A. R., Jacobs C., Rozsa E. *Effective Conversational AI: Chatbots that work*. Shelter Island: Manning Publications, 2024.
5. Infante R. *AI Apps with LangChain*. Shelter Island: Manning Publications, 2024.
6. Kimothi A. *A Simple Guide to Retrieval Augmented Generation*. Shelter Island: Manning Publications, 2024.
7. Kimothi A. *A Taxonomy of Retrieval Augmented Generation*, 2024.
8. LangChain Documentation. *Introduction*, https://python.langchain.com/docs/introduction/ (Last Access 2025/04/27).
9. LangChain Documentation. *Why LangChain?*, https://python.langchain.com/docs/concepts/why_langchain/ (Last Access 2025/04/27).
10. *LangChain*, https://langchain.com/ (Last Access 2025/03/19).

11. Lo Duca A. *Data Storytelling with Altair and AI*. Shelter Island: Manning Publications, 2024, p. 195-201.
12. Mansurova M. *Topic Modelling in production*. 2023, https://towardsdatascience.com/topic-modelling-in-production-e3b3e99e4fca/ (Last Access 2025/04/27).
13. Martineau K. *What is retrieval-augmented generation?*. 2023, https://research.ibm.com/blog/retrieval-augmented-generation-RAG. (Last Access 2025/04/24).
14. Noor F. *Seven Stage Fine-Tuning Pipeline for LLM*. 2024, https://medium.com/@noorfatimaafzalbutt/seven-stage-fine-tuning-pipeline-for-llm-ba16532cce65 (Last Access 2025/04/28).
15. OpenAI. *ChatGPT,* https://openai.com/chatgpt/overview/ (Last Access 2025/07/04).
16. OpenAI Documentation. *Fine-tuning*, https://platform.openai.com/docs/guides/fine-tuning (Last Access 2025/04/28).
17. OpenAI. *GPT-4o System Card*. OpenAI, 2024, https://openai.com/index/gpt-4o-system-card/ (Last Access 2025/05/18).
18. Pallets. *Flask Documentation*. https://flask.palletsprojects.com/en/stable/ (Last Access 2025/05/18).
19. *Pandas*, https://pandas.pydata.org/ (Last Access 2025/05/21).
20. *Python-docx*, https://python-docx.readthedocs.io/en/latest/ (Last Access 2025/03/19)
21. *Scipy*, https://scipy.org/ (Last Access 2025/05/21).