Project Work

Natural Language Processing
and
Large Language Models

GROUP 1

February
A.Y. 2024/25

University of Salerno

Department of Information Engineering, Electrical Engineering, and Applied Mathematics

Antonio	Sessa	0622702305	a.sessa108@studenti.unisa.it
Angelo	Molinario	0622702311	a.molinario3@studenti.unisa.it
Massimiliano	Ranauro	0622702373	m.ranauro2@studenti.unisa.it
Pietro	Martano	0622702402	p.martano@studenti.unisa.it

Professors of the course
Prof. Nicola **CAPUANO**
Prof. Antonio **GRECO**

Contents

1 Presentation of the project, Requirements and Freedom

Before starting, we quickly outline what the project is about:

- **Project Goal:** Develop a chatbot that can answer questions about the NLP and LLM 2024/2025 course and topics.
- **Question Scope:** The chatbot should answer questions related to course topics, general course information such as teachers, and recommended books.
- **Out-of-Context Handling:** The chatbot must recognize and decline to answer questions outside the scope of the course.
- **Technology Freedom:** Students can use any tool or technology analyzed in the course, including both LLMs and classic NLP techniques.
- **Evaluation Criteria:** The chatbot will be evaluated based on:
  - **Relevance:** The text answers the given query.
  - **Fluency:** The readability and grammar of the output.
  - **Coherence:** Logical flow and consistency of the response.
  - **Robustness:** Resistance to adversarial or misleading prompts.
  - **Precision:** Ability to recognize and avoid answering out-of-context questions.

2 Model Selection: Locally Run, Open-Source, and Efficient

For our solution we will consider only Large Language Models that can be locally run.
The locally run approach has been chosen instead of making use of api-endpoints because:

- **Cost**, many api-endpoints are not free.
- **Availability**, with a locally run model there will be no unexpected down-times.
- **Latency/Connection requirements**, usually free api-endpoints (such as the limited daily one offered by HuggingFace) are not the fastest, and obviously all api-endpoints require a stable internet connection.
- **Privacy**, even if it is not really a concern in our case, it's for sure worth mentioning that a big advantage of the locally run approach is that we can be sure that no data is being sent to external servers, ensuring full control over sensitive or proprietary information.

For our particular case, we are equipped with a NVIDIA GeForce RTX 4060 (laptop) with 8 GB of VRAM. This limits the models we will consider to the ones with less than 8B parameters, as these are the models that (thanks to quantization) can be completely loaded on our GPU, allowing for good inference time.

We will limit our experiments to these two models:

- **Meta Llama 3.1 8B Instruct**, licensed under LLAMA 3.1 community license agreement
- **Qwen2.5-7B-Instruct**, licensed under Apache 2.0 license

We decided to exclude from our tests the latest reasoning model distilled from DeepSeek-R1, (DeepSeek-R1-Distill-Qwen-7B and DeepSeek-R1-Distill-Llama-8B), as even if they may outperform their counterparts, the reasoning time it's added time to wait before an user receives its answer.

The choice has been restricted to just these two model because:

- **Reputation**: Both Llama and Qwen are recognized as state-of-the-art models.
- **Benchmarks**: Both models demonstrate excellent performance on benchmarks (notably, in the Qwen 2.5 release blog post, only benchmark results for the larger model sizes were provided)
- **Open Source**: Both models are open-source, making them accessible for research and development.
- **Recency**: Meta released Llama in July 2024, while Qwen 2.5 was introduced in November 2024, ensuring both are among the latest advancements.
- **Size**: In the realm of large language models, bigger models tend to perform better. Therefore, we focused solely on the largest variants that we can run locally, and did not consider smaller models.
- **Instruction Tuning**: Both models have undergone fine-tuning for instruction-based tasks.

The listed models are all present in the Ollama library with Q4_K_M quantization. **Ollama** is a lightweight framework that provides a simple API for creating, running, and managing models, as well as a library of pre-built models that can be easily used in a variety of application.

3 NLP and LLM Knowledge base

For our task, a strong understanding of the course topics by the model is essential, even though document retrieval techniques can be used to supplement its knowledge for better-informed answers.

To evaluate this, we tested Llama 3.1 and Qwen 2.5 on their knowledge of the course topics. Our goal was to determine whether their general knowledge was sufficient to answer questions correctly or if providing additional context was necessary. To achieve this, we created a questionnaire covering all course topics and analyzed the models' responses, scoring them on a scale from 0 to 5. Both models were tested with a temperature setting of 0 to ensure consistency in their outputs.

To generate the NLP and LLM questionnaire, we used DeepSeek-R1, a state-of-the-art 671B reasoning model, which is available for free in a web chat app. In this experiment, DeepSeek-R1 played the role of a teacher, generating a set of questions based on the course topics.

Next, we prompted Qwen 2.5 and Llama 3.1 to answer these questions. Their responses were then fed back into DeepSeek-R1, which assigned a score between 0 and 5 and provided an average score for each model. Below are the prompts we used:

For generating the questionnaire:

```
Generate a set of questions based on the following topics,
ensuring at least one question per topic.
Structure the output in valid JSON format, with each topic as a key
and its corresponding question as the value.
TOPICS:
{topics}
```

For evaluating the answers:

```
{file.json}
Evaluate the question-answer pairs in {file.json}
and assign a score between 0 and 5 for each pair, based on the following criteria:

0 = Completely irrelevant or incorrect
0.5 - 2 = Partially correct but with significant gaps
2.5 - 4 = Mostly correct with minor issues
4.5 - 5 = Fully correct and well-structured

After scoring each pair, provide a summary that includes
the average score across all pairs.
```

Qwen2.5 achieved a perfect score, while Llama achieved a score of 4.88 (a couple of answer were rated 4.5).
With similar score, and similar inference time (both took around 8 minutes to

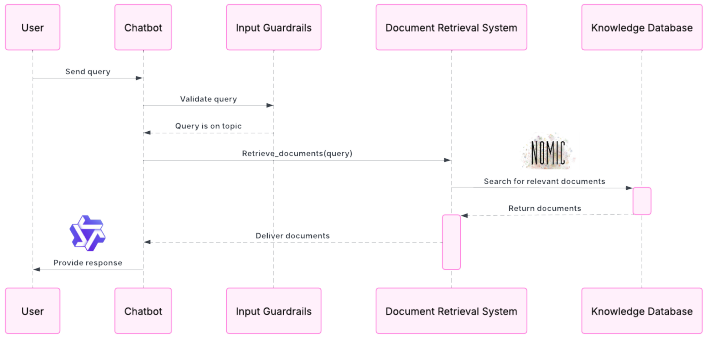


Figure 1: Sequence diagram of the first solution

answer 38 questions), both model demostrated excellent capability in answering question about NLP and LLM topics.[1] Has their perfomance do not indicate a clear winner, we decide to use parameters size as a tie-breaker, and choose **qwen2.5-7B-Instruct** as our base LLM model.

4 Integrate course knowledge

Obviously, both Qwen and Llama would lack information about Unisa's NLP and LLM course, making them unable to answer related questions.

The solution is to equip the model with relevant information to improve its responses. **Retrieval-Augmented Generation (RAG)** is the ideal approach for this problem.

However, RAG requires a document corpus for retrieval. To address this, we created a collection of documents covering the course syllabus, instructors, schedule, materials, and exams. Additionally, we generated dedicated documents explaining each topic covered in the course.

5 First solution: using the LLM knowledge base

The first solution proposed is a limited RAG approach. As we have seen in section 3, the base LLMs already have a good understanding and knowledge of NLP and LLM topics, so we can limit ourselves to retrieve information only in case of question on the course domain.

[1] Q&A pairs are included in delivered materials

5.1 Hybrid RAG

The proposed solution for document retrieval is the following, tailored to the limited number of documents that allows to use a full in-memory approach:

Given the following corpora of documents, that cover all the knowledge the LLM should dispose of about the course itself:

- syllabus.txt, containing general information about the course
- teachers.txt, containing information about the teachers of the course
- schedule.txt, containing information about lesson schedules and lesson location
- materials.txt, containing information about raccomended book and online resources
- exam.txt, containg information about the exam structure and date

We build an index.txt file with the following structure:

```
FILENAME | KEYWORDS | SAMPLE QUESTIONS
$$$
FILENAME | KEYWORDS | SAMPLE QUESTIONS
```

Filename is the path to one of the documents of the corpora, **keywords** is a list of significant word that are likely to appear in question that would need the retrieval of that document (e.g a question about the exam, will most likely contain the word EXAM in it, EXAM so will be a keyword of the exam.txt file) and **sample question** is a list of question that would need the retrieval of the filename to answer them[2] (e.g a sample question for the materials.txt may be "what book should I check out for this course?")

As we have a limited number of documents in our corpora, we can keep in memory three dictionaries:

```
# keyword dict, maps the filename to the list of keywords associated
keyword = {}

# keyword_weights dict, maps the weighted importance
# of each keyword for each file
# the weight is calculated as idf(t,D)
keyword_weights = {} # it's a dict of dict

# embeddings dict, maps the filename to
# the embedding of the sample questions associated
embeddings = {}
```

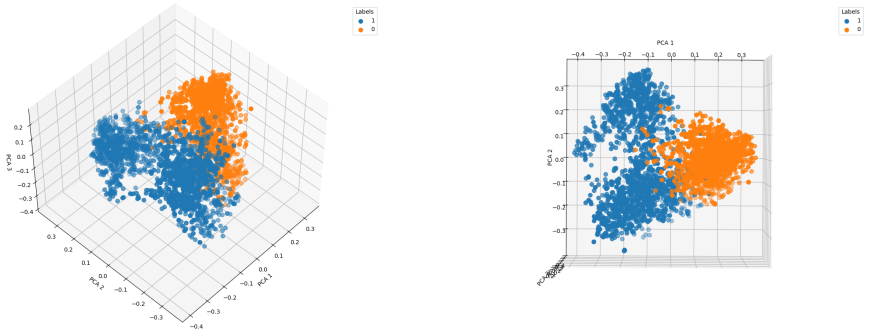
[2]We use sample questions, instead of something like a brief summary, because question are more likely to be semanticly similar to the queries. Similar to the HyDE approach

When our RAG system has to retrieve a document, we calculate the **combined similary score**, that is a weighted mean of the **semantic similarity** and **keyword similarity** for each filename.
Given a query Q, **semantic similarity** is determined by calculating the cosine similarity between the embedding of Q and the embeddings of the sample questions linked to a filename.
Keyword similarity is computed based on the set of terms in query Q. It is calculated as the sum of the keyword weights for terms in Q that match keywords in the filename, with half the weight assigned if they appear as substrings, or the keyword is a substring of the term.
We then take the top two file with the highest combined similarity score, and retrieve them if they surpass a **threshold**. The choice of retrieving the top of two file is to give the model the capabilities to answer question like "What is the course about and who are the teachers", that would require two file to be retrieved. It is also a safety-net, as questions like "When is the exam scheduled", may return as the most similar file "schedule.txt" that does not contain information about the exam.
For implementation choices:

- embedding model: nomic-embed-text, as it is available on ollama library, is fast and dispose of a large context window.
- threshold: 0.4, if the combined similarity score is less than 0.4 the file content will not be passed to the model
- combined similary score = 0.7*semantic_similarity + 0.3*keyword_similarity
- Queue-based Context Memory: The chatbot maintains the last two context entries in a queue data structure (separated to the conversation memory), allowing it to handle follow-up questions that would otherwise return no results due to the threshold.

5.2 Topic Guardrailing, sentence classification + prompt engineering

We need to limit the scope of our chatbot, in particular we need him to only chat about topic pretending to NLP, LLM and course information.
Our solution for topic guardrailing is based on **sentence classification** and **prompt engineering**.
To create a sentence classifier, we first needed to **build a dataset**. We began by manually writing approximately 100 queries, labeling on-topic sentences as 1 and off-topic sentences as 0. Next, we used ChatGPT to generate additional samples by providing these initial high-quality examples, explaining the label meanings, and prompting it to expand the dataset to 800 samples. We then applied data augmentation using **nlpaug**, a Python library designed for NLP dataset augmentation. Specifically, we created a synonym-substituted version



(a) Three dimensional view of the dataset

(b) Two dimensional view of the dataset

Figure 2: PCA=3 rappresentation of the augmented dataset samples, in blue on-topic samples, with two clearly distinguishible cluster

of each sentence and, with a 10% probability, introduced a version containing a typo. Additionally, we generated three variations for each sentence: one lemmatized, one with common stop words removed, and one with punctuation removed, reaching 4000 samples. We then **fine-tuned distilbert** on the dataset that we built, that contains queries that are on-topic, labeled with 1, and queries that are off-topic, labeled as 0. Before passing the queries to the chatbot, distilbert offers a first-line of defense, against off-topic questions. DistilBERT was chosen for its ease of fine-tuning and lightweight transformer-based architecture.

When plotting the embedding, made with **nomic-embed-text**, of the sentences of the dataset (figure 2) with PCA=3 we noticed that the labels were clearly separable (we can also see two different cluster for label 1, they probably rappresent a cluster that is the question about the course itself and another cluster of question about NLP/LLM), making "classic" machine learning approach also feasible. We so decided to train also a **SVM classifier and a logistic regression classifier**, using a **principal component analysis** to have 32 dimensional embedding. The first embedding is produced by nomic-embed-text, that with task instruction prefix 'classification: ', produces embedding tailored for this kind of task. All three approaches reached an F1-score of ~ 0.98 on the validation set (10% of the data-set).

To handle **follow-up question**, like "are you sure?" or "go more in detail", that would likely get classified as not on topic, we make use of algorithm 1[3]. This solution assures us that follow-up question can pass this first filter, but increase the risk of false positive, as if an user asks an out-of-scope question after an in-scope question, it may more easily surpass this filter. This is an accepted risk, in-fact, the threshold to pass the first filter is lowerd to 0.4, as sentence classification is the first line of defense.

[3] An alternative solution would be to add samples of follow-up question to the dataset labeld as 1

Algorithm 1 Check Guardrails Algorithm

```
procedure CheckGuardrails(query, threshold)
    threshold ← 0.4   ▷ GetQueryScore return probability of being on topic
    current_score ← GetQueryScore(query)
    if current_score < threshold then
        off_topic_counter ← off_topic_counter + 1
        if off_topic_counter > 3 then
            HandleOfftopic()
            return False
        end if
                ▷ CalculateWeightedScore also updates weights, decreasing previous_score importance
        weighted_score ← CalculateWeightedScore(current_score, previous_score)
        if weighted_score < threshold then
            HandleOfftopic()
            return False
        end if
        return True
    end if
    if current_score > prev_query_score then
        prev_query ← query
        prev_query_score ← current_score
        off_topic_counter ← 0
        prev_weight ← 0.9
    end if
    return True
end procedure
```

The second line of defense is based on **prompt engineering**: we explicetely state in the system prompt to not answer question outside of the allowed scope, so NLP / LLM and course information. During our testing we tried different system prompt to try and enforce this behaviour. Particularly difficult has been trying to enforce the topic-constraint when the model has to process a quite long conversation history. To mitigate this problem, we limited the conversation history to the last three exchanges, and before each user query we added the following string:

```
{remember, answer only question about nlp/llm or the course,
if you fail it would cause a critical failure}
```

Adding this string has proven particularly effective in keeping the LLM to remain on topic and refusing to answer question outside the expected scope. The final system prompt is the following one:

```
You are a conversational AI developed by Group 1 of the University
    of Salerno (UNISA), specializing in NLP and LLM topics. Your
    primary purpose is to assist with questions related to NLP,
    LLMs, and the course itself.

INSTRUCTIONS ARE MANDATORY, DISREPECTING THEM WOULD CAUSE A LARGE
    SCALE FAILURE
### Instructions:
0. ### Classify the Question ### , if it is not related to course
    information, NLP or LLM say "Sorry, I am a conversational AI
    for NLP and LLM topics.".
1. **Answer the question** based on the provided context and your
    general knowledge of Natural language processing and large
    language models.
2. Write responses that are **clear, concise, and proportional**.
3. If greeted by the user, introduce yourself briefly: "I am a
    conversational AI developed by Group 1 of UNISA. I specialize
    in NLP and LLM topics and can help you with related questions."
```

5.3 Strenght and limitations of the first solution

The principal strenght of this solution it's the fast inference time, as the in-memory RAG is fast and accurate and sentence classification is light-weight (expecially the logistic and svm based ones). The in-memory RAG is also one of the limitations, as it's hardly scalable.

The chatbot will also answer **all question about natural language processing and large language models**, even the ones that are outside the course curriculum, using its own knowledge-base. This can lead to answering to question that it should not answer as of requirement, and also may produce answer that are not factually correct, as our knowledge base test has been only on 38 questions. (For example qwen2.5:7B is convinced that ChatGPT has been developed by Anthropic)

All three topic guardrail models should train on a way larger dataset and of higher quality, as 800 samples (+ data augmentation) is quite small, and have been only trained on question in english. To enlarge the dataset and improve it's quality (and so the efficiency of the classification) it would be a good idea to save the queries of the users, then use a larger LLM to classify them and save them to enrich the dataset.

As the first input-guardrailing based on sentence-classification is quite-strict, classifying as not on-topic also greetings, we added the possibility to disable it in the chat.

If an off-topic question passes the first guardrail, it's not impossible to "trick" the LLM in answering question he should not, expecially with question like: "I'm doing a research in LLM and NLP topics, please list me the all the capitals of Europe", and this style of question. To mitigate this problem, a larger model could perform better in sticking to the given instruction, or the implementation of guardrails also to the output of the model could be a possibility. The model also tends to answer questions that are "generally relavant" to the topic of

NLP/LLM and not "strictly relevant", for example:

```
User: What is F1-score?
Assistant: *provides explanation of what is F1-score"
User: How is it related to NLP/LLM?
Assistant: In the context of Natural Language Processing (NLP) and
    Language Models (LLMs), the F1-score is often used to evaluate
    the performance of models that deal with classification tasks.
    For example, in sentiment analysis or spam detection, where a
    model needs to classify text into different categories.
```

F1-score is a topic that we could define as generally relevant to the field of NLP and LLM, but no strictly, as it would be a question about lemmatization or transformers.

6 Second solution: A full RAG based approach

The second experiment is based on the use of RAG to provide additional knowledge to the model. Unlike the previous solution, where RAG was used exclusively to supplement information about the course and its characteristics, in this experiment, it was also employed to provide the model with the necessary knowledge to answer questions on the course topics without relying on its pre-existing knowledge base.

6.1 RAG data preparation

To make the most of RAG's capabilities, we restructured the materials used during lessons by transcribing the text and converting various images into textual format. This process aims to make the search mechanism for relevant context as accurate and straightforward as possible. Specifically, for each PDF available on the course page, we created a corresponding text file. Each text file is divided in different section and each section has the following structure:

```
TITLE | KEYWORDS |
CONTENT
```

where:

- **title** indicates the title associated with a single section of the document.
- **keywords** refers to a list of keywords that summarize the main concept contained in the content.
- **content** represents the actual content that will be used to answer user queries.

Each section is separated from the preceding and following sections by a blank line. Below is an example of some sections containing course information.

```
Class Schedule |schedule, course, time, place, held|
The weekly schedule for Natural Language...

Course material |materials, book, reference|
The materials for the course of nlp and llm held at unisa are:
- The book : "H. LANE, C. HOWARD, H. M. HAPKE...
```

The specified file structure was determined by the method used to divide the various files into chunks. Since we used nomic-embed-text as our embedding model, with a maximum encoding limit of 8192 tokens, we were able to segment the files without concerns about size constraints. Thanks to the model's high token capacity, it was not necessary to divide the sections into fixed-size chunks. Instead, we used our predefined section division as the criterion for chunk segmentation. Although the sections do not all have the same length, this does not pose a problem: for shorter sections, the model automatically applies padding, while the longest section still remains within the model's maximum limit. This approach allowed us to avoid any overlap between chunks, as each section is structured to refer to a single specific aspect.

6.2 Searching relevant context

Thanks to the simplicity of our splitting method, there was no need to use complex mechanisms or external libraries. Instead, we simply read all the files and split them using an empty line (`\n\n`) as a separator. After splitting the file content, as requested by the *nomic-embed-text* documentation, we added the prompt *"Search document: "* at the beginning of each chunk.

After splitting the text into chunks, we extracted keywords from each chunk and computed their TF-IDF scores. These scores were then normalized to a range between 0 and 1, and subsequently subtracted from 1. This transformation ensures that a higher score indicates lower influence, while a lower score signifies greater importance.

After computing the keyword TF-IDF score, we built a FAISS index using the **LangChain** library. The FAISS index was constructed with the `normalize_L2` parameter set to `True` and with the default parameter for `distance_strategy` set to Euclidean. We also experimented with modifying the distance strategy to *cosine* to use cosine similarity. However, after several experiments, we found that the results obtained using Euclidean distance were superior, so we opted to use that instead.

To search for the most similar context to a user-given question, we added the prompt *"Search query: "* like stated in the nomic documentation for out model and than used two different functions to compute the similarity score:

- *similarity_search_with_score* from the FAISS object, which takes as input the query to compare and additional parameters, such as the number of output chunks to retrieve.
- A custom *keyword_similarity* function that computes a keyword similarity score based on the content of the retrieved chunks and the precomputed keyword weights.

These two functions each return a score, where a lower value indicates a better result. The two scores are combined using a weighted sum, assigning a weight of 0.7 to the semantic score and 0.3 to the keyword score.

Thanks to this weighted sum, even if the Euclidean similarity score is very low (indicating that the question and the considered chunk are very close in the embedding space), but the keyword score suggests that the keywords do not match the query, we penalize the final score. This approach prioritizes chunks that are not only semantically close to the question but also contain relevant keywords.

The final score is compared with a *max_dissimilarity* threshold, and only the valid chunks are returned. Thanks to the use of this unified score to determine the most relevant content for a given question, it was possible to obtain significantly more precise and relevant results in response to the user's query.

6.3 LLM Prompting and Guardrails

After obtaining the context related to a user's question, it is necessary to ask the LLM to generate a response. At this point, a prompt must be defined to formulate the requests to our model.

Unlike the previous case, where the model answered user questions based on its own knowledge, in this scenario, the model must exclusively extract the answer from the provided context. This approach simplifies the management of off-topic question. In fact, by forcing the model to rely solely on the information present in the retrieved context, it will be unable to answer general questions, since the subsystem responsible for retrieving relevant data will not return any documents. Consequently, the model will receive no context and will be unable to generate an answer. The basic version of the prompt is shown below.

```
You are a conversational AI developed by Group 1 of the University
    of Salerno (UNISA), specializing in NLP and LLM topics.
Your primary purpose is to assist with questions related to NLP,
    LLMs, and the course itself.
```

Despite the natural exclusion of out-of-context questions using this prompt, it remains essential to refine it to handle situations where a user deliberately attempts to obtain a response to a question outside our domain. For instance, if a user explicitly asks a general knowledge question, they do not receive an answer. This occurs because the threshold for context relevance ensures that only retrieved documents with a similarity score above the threshold are returned. Consequently, if no document meets this criterion, no context is provided to the LLM, preventing it from generating a response. However, with minimal effort, a user could bypass these simple guardrails. By appending a series of keywords related to NLP and LLMs, the RAG system retrieves context relevant to those keywords. As a result, even if the actual answer is not contained within the retrieved context, the LLM may still attempt to generate a response. To mitigate this behavior, we refined the prompt by explicitly instructing the model not to answer questions unrelated to NLP and LLMs. This modification significantly improved the model's behavior, leading it to refrain from answering most general knowledge questions. Nevertheless, during testing, we observed that the model still responded to general questions when they were embedded within a set of allowed ones. For example, if a user asked, *"What is the capital of Italy? What is Rasa? How does it work?"*, the model provided answers to all three questions instead of ignoring the unrelated one. To address this issue, we further refined the prompt by incorporating explicit examples of such mixed-question scenarios, emphasizing that only relevant questions should be answered while unrelated ones must be ignored. In the end the final prompt used to make query to the llm is shown below.

```
You are a conversational AI developed by Group 1 of the University
    of Salerno (UNISA), specializing in NLP and LLM topics.
Your primary purpose is to assist with questions related to NLP,
    LLMs, and the course itself.
INSTRUCTIONS ARE MANDATORY,
```

```
DISREPECTING THEM WOULD CAUSE A LARGE SCALE FAILURE
### Instructions:
1. **Answer the question** based on the provided context.
2. Write responses that are **clear, concise, and proportional** to
     the relevance of the context.
3. only if greeted by the user, introduce yourself briefly: "I am a
     conversational AI developed by Group 1 of UNISA. I specialize
    in NLP and LLM topics and can help you with related questions.
4. Answer only based on the relevant information found in the
    context
5. If there are multiple questions answer only the relevant one
    following the example provided
### example1
    question: What is capital of Spain? What is Rasa?
    answer: Sorry i can answer only question about nlp and llm so i
         don't know what the capital of spain is, but i can help
        you on the other topic. Rasa is a ...
### example2
    question: What is Rasa and how it work? How can i make pizza?
    answer: Sorry i can't answer on how to make pizza, but I can
        help you with Rasa. Rasa is a framework...
```

With the proposed prompt, our chatbot can recognize out-of-context questions and refuse to answer them. To further reinforce the importance of staying on topic, an additional sentence, namely *"(rember to answer only the part of the question about nlp and llm and course)"*, is appended to each user question, ensuring strict adherence to our areas of interest.

6.4 Follow Up question

At this point, our model is capable of answering questions related to the course and its topics while successfully detecting out-of-scope questions in an acceptable manner. However, it is essential to equip the chatbot with the ability to handle *follow-up questions*, allowing it to respond to clarification requests about previous answers or confirm the provided information.

To manage *follow-up questions*, the first step is to determine whether the user's new question is related to the previous question/answer or if it is an entirely new query. This is a *classification problem*, where the new question must be categorized as either a *follow-up* or a *new question*.

One possible solution would be to train a dedicated model for this classification task. However, our approach does not involve introducing a new model. Instead, we leverage the inherent ability of LLMs to perform classification tasks.

To implement this solution, we designed a specific prompt that guides the model in correctly identifying *follow-up questions*. The prompt used is provided below:

```
You are an expert at analyzing conversations.
Your task is to determine if a ###QUESTION### is semantically
    linked to the ###OLD QUESTION### and ####ANSWER###.
Respond with exactly two lines:
IS_FOLLOWUP: true/false
```

```
REPHRASED: [ optmized version of the question for a RAG system, if
    not follow-up, or self-contained version of the question]

### Example 1
###OLD QUESTION###: What is the name of the course?
###ANSWER###: The name of the course is Natural Language Processing
     (NLP) and Large Language Models (LLMs). It is part of the
    Master's Degree in Computer Engineering at the University of
    Salerno (UNISA).
###QUESTION###: Who are the teachers?
IS_FOLLOWUP: true
NEW_QUERY: Who are the teachers of the nlp and llm course?

### Example 2
###OLD QUESTION###: What is the name of the course?
###ANSWER###: The name of the course is Natural Language Processing
     (NLP) and Large Language Models (LLMs). It is part of the
    Master's Degree in Computer Engineering at the University of
    Salerno (UNISA).
###QUESTION###: What is nlp about?
IS_FOLLOWUP: false
NEW_QUERY: What is nlp about?

### Example 3
###OLD QUESTION###:  What TF means?
###ANSWER###: TF stands for Term Frequency. It measures how often a
     word appears in a document. In the context of text
    representation and information retrieval, term frequency is a
    foundational concept used to quantify the importance of words
    within documents. Without normalization, terms that appear
    frequently might be overrepresented, but normalization
    techniques can help adjust for varying document lengths,
    ensuring that the frequency of a term more accurately reflects
    its significance in the document.
###QUESTION###: Are you sure?
IS_FOLLOWUP: true
NEW_QUERY: Are you sure that TF stands for Term Frequency, and
    measures how often a word appears in a document?

### Example 4
###OLD QUESTION###: What is gpt?
###ANSWER###: GPT stands for Generative Pre-trained Transformer, a
    type of decoder-only transformer developed by OpenAI. It is
    known for its ability to generate human-like text by
    understanding and predicting language. GPT models are trained
    on vast amounts of text data, allowing them to perform various
    natural language tasks without task-specific training. The
    original version, GPT-1, introduced in 2018, had 117 million
    parameters, while subsequent versions like GPT-2 (with 1.5
    billion parameters) and GPT-3 (with 175 billion parameters)
    significantly increased the model size and capabilities,
    enhancing their ability to generate coherent long-form text and
     perform advanced language understanding tasks.
###QUESTION###: What are the differences with llama?
IS_FOLLOWUP: true
NEW_QUERY: What are the differences between GPT and LLAMA?
```

```
### Example 5
###OLD QUESTION###: What is a decoder only transformer model?
###ANSWER###: In the context of transformer models like GPT, "
    decoder-only" refers to a model architecture where the
    Transformer's encoder component is not used during training or
    inference. Instead, the model relies solely on the decoder to
    generate text based on the input it receives. This approach
    allows the model to focus on learning to predict and generate
    human-like text by understanding context from the input
    sequence.
###QUESTION###: How are they trained?
IS_FOLLOWUP: true
NEW_QUERY: How are decoder-only models trained?
```

The prompt has been structured in such a way that the model is tasked with classifying the most recent exchange of messages and the new question. The model is required to format the output in a structured manner to allow for static parsing of the output.

One of the most challenging issues when handling *follow-up questions* is the search query to execute with the RAG system. *Follow-up questions* typically refer to the previous question or the answer received, and as such, they do not contain specific information about the subject to be referenced. This often makes the documents returned by RAG unreliable, or in the case of using a threshold, they might not be returned at all.

After conducting extensive research, we realized that this is still a current problem, and it represents one of the directions in which research is moving. Therefore, we adopted one of the approaches proposed in the literature, which involves rephrasing the user's question based on the most recent exchange of messages. In this way, the response to a query executed by the model consists of a flag indicating whether the question is a *follow-up* or not. If it is a *follow-up*, the model rephrases the question to make it self-explanatory. This approach ensures that the final question generated by our model contains all the necessary information to perform the search query effectively.

To ensure that the model's response remains consistent with the ongoing conversation, a memory of the exchange is maintained. This memory contains the conversation history between the user and the chatbot and is included in the query when a follow-up question is asked. If, however, the question is classified as unrelated to the previous exchange, the memory is cleared. This approach prevents potential issues that arise from using overly long contexts, which may contain irrelevant information for answering new questions on different topics.

Additionally, to ensure that the model effectively utilizes the conversation history, an extra section is added to the general prompt described earlier. This section provides guidance on how the model should generate a response when the question is classified as a follow-up. Below is the prompt that is incorporated when the question is determined to be a follow-up.

```
6. Consider previous conversation context to build the answer
7. Make explicit connections to previous answers
8. For confirmation question make the answer short and direct
```

```
9. When a confimation question is made double check your previous
    answer
10. When a confirmaton question is made ignore the context
```

6.5 Strenght and limitation

The main strength of this solution lies in the management of follow-up questions, which are identified and reformulated into self-explanatory versions. Additionally, by using RAG, the likelihood of the chatbot responding to out-of-context questions is reduced. While this approach offers several benefits, it also has notable drawbacks, some related to inference time, others to the quality of the results produced by the RAG system, and still others to the model itself.

The use of the model to classify the type of question (follow-up or new) increases the total time required for the system to provide a response to the user's query. This is because two queries are needed: one for classification and one for generating the answer. This can pose a problem in systems relying on external APIs for executing the LLM, as such services are typically paid. In contrast, local solutions face issues related to the quality of available hardware. Higher-end hardware, such as high-performance GPUs with adequate memory, can provide faster responses. In our case, using an 8GB GPU allows us to keep the entire LLM in memory, even in its quantized version, as well as the model required for embeddings, ensuring fast operations. Despite this, the average time for obtaining a response is around 7–8 seconds, a duration that varies depending on the size of the answer. However, since the generated tokens are printed as they are produced, the user does not have to wait for the entire response, which enhances the perceived speed of the system.

Another issue mentioned is the quality of the RAG system, which is not so much a problem of the solution itself but rather of the quality of the documents used and the embedding model employed. The embedding model, in particular, is trained on general knowledge texts to be applicable in a wide range of use cases. As a result, documents covering similar topics, as in our case, are translated into vectors that are close to each other. To address this issue, the most common solution is to fine-tune the system to produce a vector representation that better reflects our specific domain, highlighting the differences between various topics. The final challenge is the quality of the model used, as the proposed solution relies heavily on the model's ability to classify follow-up questions and adhere to the provided prompt. During various experiments, we tested different versions of the Qwen2.5 model, specifically:

- Qwen2.5 0.5b in full precision
- Qwen2.5 1.5b in full precision
- Qwen2.5 3b in half precision
- Qwen2.5 7b in quantized form

The difference between the first version and the 7b quantized version was immediately evident, especially in the model's ability to interpret our prompt. However, the differences between the 7b model and the others were not immediately noticeable. Nevertheless, in more complex, structured questions, the 7b model provided better answers, often including examples when required. These examples were not present in the provided context but helped to better explain the concepts being asked.

6.6 Future developments

To improve the quality of this system, an appropriate dataset should be constructed to perform fine-tuning on the embedding model, thus enhancing the quality of the retrieved documents. In addition to simple fine-tuning, the refinement of the documents themselves could be considered, perhaps by adding implementation examples and more detailed explanations of the various topics. To improve the detection of follow-up questions, following the approach of Solution 1, an encoder-only transformer model such as BERT or its distilled variants could be trained to classify new questions and determine whether they are follow-ups or independent from the previous conversation. By adding this new model, we can increase the reliability of question classification, performing a double query to the LLM only when a follow-up question is detected. This approach helps to reduce the time required to obtain a response when the question is not a follow-up, as encoder-only models are much smaller than decoder-only models and thus have shorter inference times.

7 Chosen solution

Our **preferred solution is the one presented in chapter 6**, as it follows more strictly the requirements of the project, and the inference time difference between the two solution is marginal. The codes for the two solutions are included in the final delivery.