Specific Chat Bot with RAG

Final project as part of the CAS NLP carried out by the
Mathematics Institute of the University of Bern

Marc Eyer

June, 14th 2024

contact: marc.eyer@phbern.ch

Contents

1 Abstract

This report documents the final project of the CAS Natural Language Processing organized by the Mathematical Institut of the University of Bern. The aim of the final project was to set up a domain-specific chat bot that works according to the RAG (Retreaval Augmentes Generator) principle. This means that the automated answering of customer questions by a language generator is based on domain-specific documents. The customer question and the relevant context are then passed together to the prompt of the language generator, which formulates an answer based on the specific documents. Today, the RAG technique is one of the standard techniques for the task of "abstractive" question answering. In contrast to the "extractive" Q&A, the "abstractive" Q&A aims to generate an answer from the context that correctly answers the question, where the "extractive" Q&A just extract the answer from the given context without generating a new text. In this work, we succeeded in building a functioning RAG-based chat bot based on 38 domain-specific documents. These are basic documents of the Bern University of Teacher Education (legal bases such as guidelines, regulations and study plans). The results of the chat bot were benchmarked against a collection of 1030 manually generated question and answer pairs with acceptable results. For the deployment a prototype of a streamlit app was built and tested.

2 Introduction

The Bern University of Teacher Education (PHBern) is a complex organisation. Around 20 different degree programmes are offered in four institutes, which are responsible for training teachers at various levels. Each degree programme follows its own curriculum, in which students are trained in three areas (educational and social sciences, subject didactics and practical vocational training). How this is done in concrete terms, which modules and learning opportunities are offered, what the exact scope and content of these are and what dependencies exist between them is regulated in detail in guidelines. It is often a great challenge for students to quickly gain an overview of the programme at the PHBern in the jungle of regulations, directives and guidelines. Accordingly, student counselling is an important element at the PHBern. The student counselling service is available to provide information of all kinds. Some of the information requested are standard questions, the answers to which could be very easily automated or recorded and dealt with in 'FAQ' catalogues. Others are complex, individual or cannot be answered unambiguously and their handling cannot be automated or standardised accordingly. However, automating the processing of standard enquiries in particular would be a great relief for the student advisory services and at the same time provide a great service to students due to the constant availability of information.

The major tech companies that offer LLMs nowadays provide APIs or even direct applications through which own documents can be uploaded, which the chat bots can then access to include specific, not generally available information about an institution or company in the generation of answers to questions. The technology behind this is called Retreaval Augmented Generation, or RAG for short.

The question is justified as to whether it is worthwhile setting up a RAG system ourselves, given that these services work very well. I decided to give it a try as part of this CAS course. Firstly and mainly to understand how it works and to learn how something like this can be built. Secondly, to be as independent as possible from the commercial providers and to have the opportunity to make customisations myself. Due to the technical possibilities and the time available, I finally decided in favour of a compromise. On the one hand, I rebuilt a retrieval mechanism, but used a pre-trained model for the generator part, without fine-tuning it, for example.

If the PHBern wants to have a powerful chat bot in the medium term that is able to represent the PHBern in detail, the question will arise again as to whether an in-house system should be built or whether commercial products should be used. I will come back to this in chapter 7.

3 Basic Principles

3.1 Q&A, a typical NLP task

Q&A (questions and answering) is one of the typical tasks that NLP attempts to solve. There are two different types of Q&A. If the aim is to find and write out an existing answer in a given text chunk, this is called *extractive* Q&A. If the aim is to formulate a new answer to a question based on a context, this is called *abstractive* Q&A (HuggingFace_1 2024).

3.2 The RAG Technique for Q&A

In order to develop a domain-specific chat bot for the PHBern, I need a technique that can solve the task of *abstractive* Q&A (see chapter 3.1). This is realised with the Retreaval Augmented Generation (RAG) technology, see i.e. (Gao 2023). The RAG technology is an efficient and very successful technique for generating an answer based on specific documents with a language model. The term *retrieval-augmented* expresses the fact that the text generation of a language model is supported by the results of a search for relevant content in specific documents. The mechanism therefore consists of two parts: firstly, the search for relevant information in an available corpus and secondly, the generation of an answer based on the information and the question posed.

The architecture of a RAG model is described in fig. 1. It consists of the following steps:

1. Receiving *a question* to be answered:

A question in the form of a text must be made available to the RAG in some way. This can be done via a prompt or via a front-end input mask.

2. Finding a suitable text passage from a specific text corpus from which the question can be answered:

The entered question text is embedded in a vector space using a language model, this is called *semantic sentence embedding*, see i.e (Niels Reimers 2019). All relevant documents on the basis of which the question is to be answered are also first embedded in the same vector space using the same language model. The documents are often prepared as a vector database for this purpose. The input text, which is now available as a vector in the vector space, is then compared with the contents of this vector database and one or more relevant entries in the vector database are assigned to the input vector (*semantic search*). In this way, a semantically relevant context from the vector database is assigned to the input text.

3. Feeding the prompt of a LLM:

With the aim of generating an output text, the question and the context text assigned to the question from step 2 are now written into the prompt of a

generative language model. The prompt also contains general instructions such as '*answer the question in one or two short German sentences*' or similar.

4. Post Processing:

In a fourth step, the output of the language model can be edited in a *post-prosessing*. For example, a plausibility check of the response could be carried out in some way before it is output to the client. This could also be done with an AI machine.

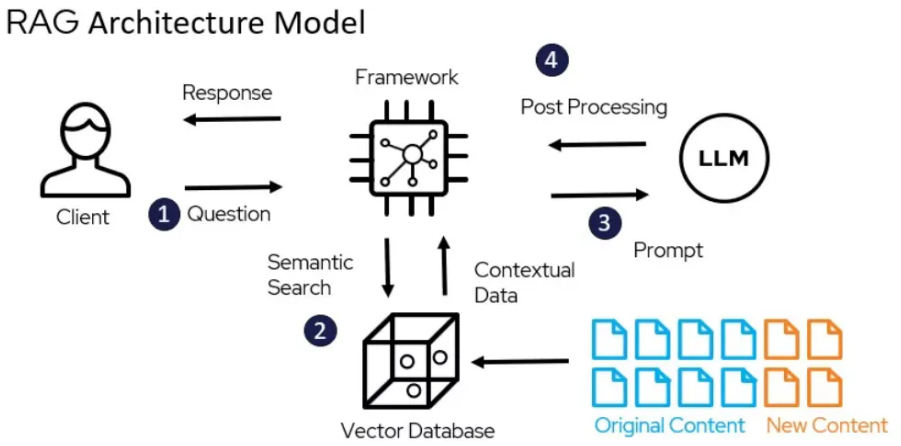

Figure 1: Shown is a schema of a RAG Architecture. The graphic was created by (Ghosh 2024)

4 Project: Building a RAG

Of course, there are different versions of the RAG architectures, simpler and more advanced or more sophisticated. To start with, I have opted for the simplest version. This is shown in the following flowchart (2) by (Gao 2023) and essentially consists of a vector database with the relevant documents, a retrieval block and the LLM as a language generator, whereby the LLM is adopted in a pre-trained version without any further fine-tuning.

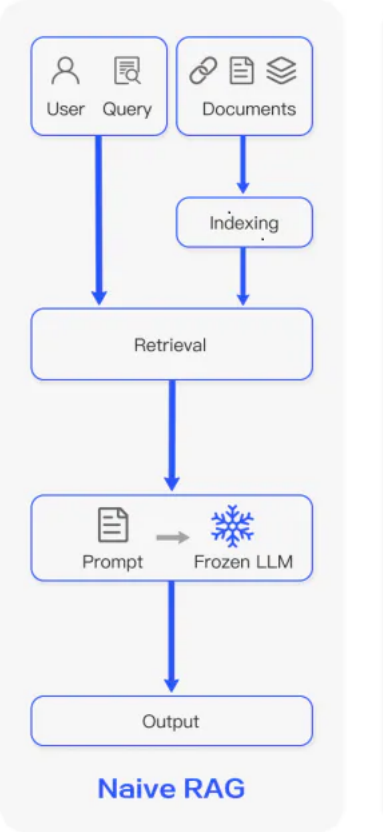

Figure 2: Simplest RAG

The planned project steps were as follows.

1. Compilation of the text corpus
2. Text analysis
3. Preprocessing the text
4. Compilation of a vectordatabase from the corpus
5. Creating a reriever
6. Load and setup the LLM
7. Creating a LLM Chain
8. Pasting the pipelines together to a RAG Chain

In the following chapters I will explain exactly how I carried out the individual steps.

4.1 Compilation of the Text Corpus

In this part, I describe the data and the text corpus in general and then show how I had to prepare it for use in the RAG.

At the beginning of the project, the question arose as to where the information that users of a chat bot at the PHBern could request could be found. In fact, only very little information is written down in documents, and only information that can be formally recorded. These are regulations, statutes, curricula, directives and other official documents that govern the organisation at the PHBern. However, a great deal of information on everyday questions from students can only be derived indirectly from the information in these formal documents. For example, the answer to the question '*Can my girlfriend help me write my module assignment?*' can be derived indirectly from the passage in a directive '*Module assignments are generally individual assignments*' combined with common sense.

One text source that contains much more non-formal information is emails from correspondence between the student counselling service and students. A systematic recording and evaluation of such questions and information would be very valuable for the processing of this issue. However, the use of such emails immediately raises the question of data protection. Even if the emails were anonymised, it is not easy to gain access to employees' email accounts.

In this study, I have therefore only integrated formal, generally accessible documents into the text corpus. It consists of 38 documents (see Appendix 1 in Chapter 9.1) that are freely accessible on the PHBern website.

4.2 Text Analysis

To get an overview of the text, I made some statistical analyses of the text. For example, it gives an overview of the richness of words and the use of very specialised words.

Before I analysed the text, I saved all the documents, which were mainly in .pdf and .docx format, as .txt files. Apart from that, the documents were hardly changed. In particular, for the main project the texts were not preprocessed (no stemming, lematisation, change of capitalisation, removal of stop-words and the like) and numbers and abbreviations were left in the text.

Later I experimented with pre-processed text (removal of stopwords and lemmatisation) and got much worse results than with the uncleaned text. I interpret this as an indication that the full information is necessary to obtain the richest possible text generation (see chapter 5.3).

4.2.1 Number of words per document

In the following figure 3 you can see that the documents used vary greatly in scope. Some contain almost 32'000 characters, the smallest just under 1000.

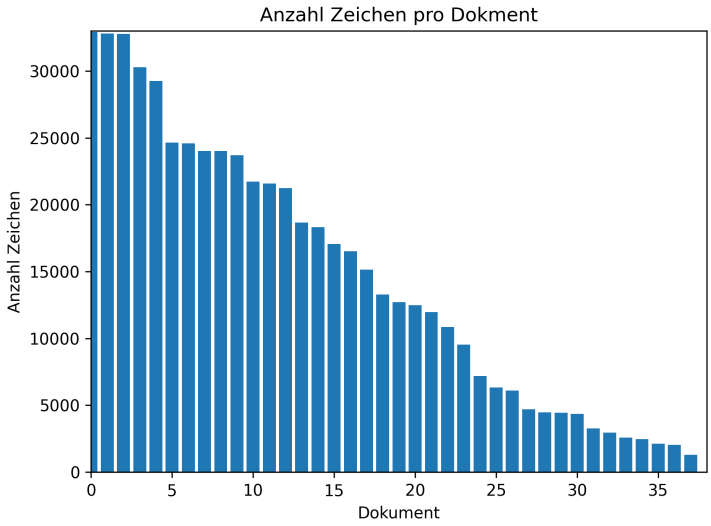

Figure 3: Number of characters per document

4.2.2 Number characters per word

The next figure 4 shows the frequency of occurrence of words as a function of word length. The dominance of 3-character words is clearly recognisable. These are the articles and the conjunctions, which have a significant impact here. Only few words with a lenth of 20 characters or more occure in the text.

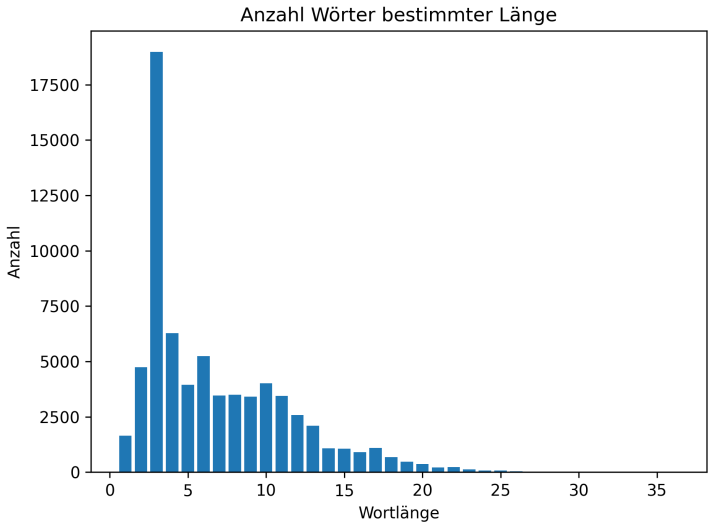

Figure 4: Number of words of a certain length

4.2.3 Occurance of words

Figure 5 shows the frequency of occurrence of words. The graphic is cut off on the right. There are many words that occur very frequently (which can no longer be seen in the graphic). However, it is interesting to note that there are many words that occur very rarely or only once. This makes the task particularly challenging for language generation and indicates that this is an expert text or a text from a domain with specialised vocabulary. However, it can also be assumed that the column of words that only occur once (or twice) contains all the miss spelt words.

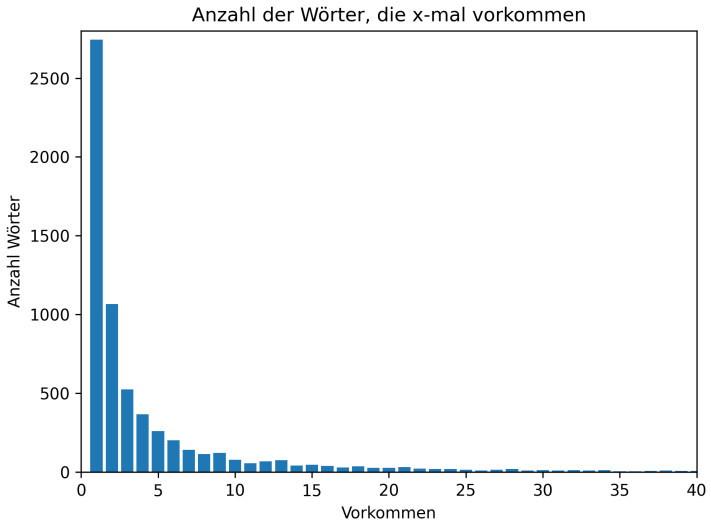

Figure 5: Occurance of words

4.2.4 Bi-Grams

Rather for fun or for a plausibility check, I have created a statistic of the most frequent bi-grams. Bi-grams are tuples of words that occur together. The results (fig. 6) seem quite plausible.

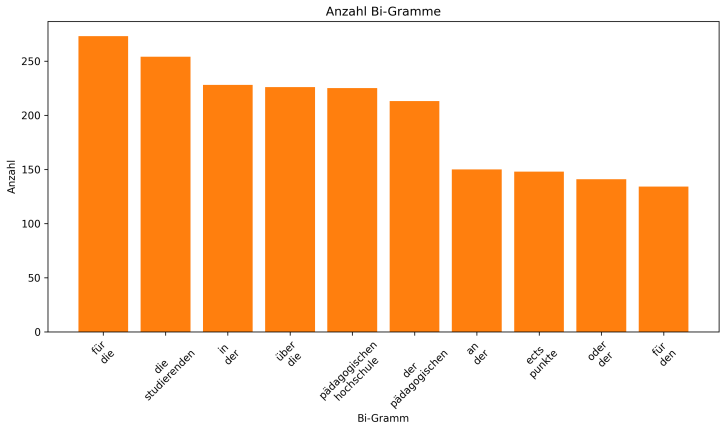

Figure 6: Most frequent Bi-Grams

4.3 Preprocessing the text

With regard to the creation of a vector database with the text corpus, I had to divide the text documents into text chunks that can be embedded in a vector space using a language model. This is called *semantic sentence embedding*. A very helpful package for this task is *LangChain* (LangChain 2024). **LangChain* provides a wide range of easy-to-integrate packages that make it easy to work with LLMs. The *LangChain* method `CharacterTextSplitter` was used in connection with this project.

First of all we define a function `text_splitting`.

```
def text_splitting(chunk_size, overlap, separator=""):
    '''
    Imports needed:
    - from langchain.text_splitter import CharacterTextSplitter

    Description:
    Splitts the text in a certain number of chunks with an
    overlap of "overlap".
    The Separator should be set to separator=""!
    The function returns a text_splitter.

    Variabels:
    - chunk_size: int; Numer of characters in the chunk
                  (i.e. 500)
    - overlap: int; Number of characters to overlap between
               the chunks (i.e. 10)

    Return: function
    '''
    from langchain.text_splitter import CharacterTextSplitter
    text_splitter = CharacterTextSplitter(
            chunk_size=chunk_size,
            chunk_overlap=overlap,
            separator= separator
            )
    return text_splitter
```

Then we need a helper-function to read the first line of all our documents, since this is the titel of each document. I call it `get_text_before_newline`:

```
def get_text_before_newline(text):
    '''
    Imports needed:
    - re

    Description:
```

```
    Helper Function to read the first Line in each Document, which
    is the title of the document! The Regular Expression is to
    match any char-sequence before the first newline character.
    The function returns a srting with the titel of the document.

    Variabels:
    - text: list of str; The input variable is a list of strings
            with the Titel of the Document in the first line
            sperated by \n.

    Return: str
    '''
   import re
   match = re.search(r'^(.*?)\n', text, re.DOTALL)

   if match:
    return match.group(1)
   else:
    return 'No title found!'
```

In order to split the text into suitable chunks, I now had to merge the whole text into a single text file. This was done with the following functions `load_text` and `create_LangChain_doc`. The first one creates a list of strings from the whole text:

```
def load_text(dict_path, N):

    '''
    Imports needed:
    - os

    Description:
    Loads ths Text files into a list. The function returns a list
    of N strings.

    Variables:
    - dict_path: str;  The path of the folder the docs are placed
    - N: int OR str; It is a integer OR the string 'all'.
         If a integer is found, the corrsponding number of
         documents are loaded. If 'all is found, all
         documents are loaded.

    Return: list of str
    '''
    import os
    dict_path = dict_path
```

```
    entries = os.listdir(dict_path)
    # Filter out entries that are files
    files = [entry for entry in entries if  \
        os.path.isfile(os.path.join(dict_path, entry))]

    N_docs_all = len(files)

    if isinstance(N, int):
        N_docs = N
        # Handle the integer case
        # if N_docs represents a number of documents to process
        # process_integer_docs(N_docs)
    elif isinstance(N, str):
        N_docs = N_docs_all

    texts =[ ]
    for i in range (1,N_docs+1):
      with open(dict_path+f'/Doc{i:02}.txt', 'r', \
        encoding = 'utf-8') as file:

        texts.append(file.read())

    return texts
```

TWith the second function I can create a *LangChain*-object, which is a document that contains a list of text chunks.

```
def create_LangChain_doc(texts, chunk_size, overlap, separator):
    '''
    Imports needed:
    - from text_loader import get_text_before_newline
      (Helper Function from above)
    - from text_loder import text_splitting

    Description:
    A langchaine-Document with the "create_documents"-methode
    from TextSplitter is created. The TextSplitter uses a list
    of texts (here: 'texts') and a list of metadatas
    (here: 'metadatas') to create a list of text chunks.
    The helperfunction "get_text_before_newline(text)"
    is used to read out the title of each document.
    The function returns a langchain-object

    Variabels:
    - texts: list of str; The input variable is a list of strings,
```

```
            with the Titel of the Document in the first line
            sperated by \n.
    - chunk_size: int; Numer of characters in the chunk
                  (i.e. 500)
    - overlap: int; Number of characters to overlap between
               the chunks (i.e. 10)

    Return: LangChain-Object
    '''
   from langchain.text_splitter import CharacterTextSplitter

   metadatas = [ ]

   for text in texts:
      metadatas.append({"document": get_text_before_newline(text)})

   documents = text_splitting(
            chunk_size,
            overlap,
            separator
            ).create_documents(texts, metadatas=metadatas)

   return documents
```

Finally this *LangChain*-object is split-up into a certain number of chunks with the function `chunk_docs`. It returns a list of *LangChain*-objects.

```
def chunk_docs(documents,chunk_size, overlap, separator):
    '''
    Imports needed:
    - from text_loder import text_splitting (text_splitter from
      CharacterTextSplitter sis used)

    Description:
    The LangChain-Object is split up into the nummer of Chunks,
    that is defined in the text_splitting function and stored in
    a List of LangChain-Objects.
    The function returns a List of LangChain-Objects.

    Variabels:
    - documents:  LangChain-Object; The input variable is a
                  LangChain-Object.
    - chunk_size: int; Numer of characters in the chunk
                  (i.e. 500)
    - overlap: int; Number of characters to overlap between
               the chunks (i.e. 10)
```

```
    Return:     List of LangChain-Objects
    '''
    from langchain.text_splitter import CharacterTextSplitter
    chunked_documents = text_splitting(
                chunk_size,
                overlap,
                separator
                ).split_documents(documents)

    return chunked_documents
```

Once all the funtions are defined we can set the parameters and execute them. Here we set the paramter `chunk-size` to 500 and we define an `overlap` of zero.

```
###################################
# Prepare Documents
###################################

#Set Chunk-Size and Overlap
chunk_size = 500
overlap = 0
dict_path = '/content/drive/My Drive/ColabNotebooks/\
    CAS_NLP_final_project/Textdata/'

texts = load_text(dict_path, 'all')

text_splitter = text_splitting(
            chunk_size,
            overlap,
            separator=""
            )

documents = create_LangChain_doc(
            texts,
            chunk_size,
            overlap,
            separator=""
            )

chunked_documents = chunk_docs(
            documents,
            chunk_size,
            overlap,
            separator=""
            )
```

4.4 Compilation of a vectordatabase from the corpus

One of the most common ways to store and search over unstructured data is to embed it and store the resulting embedding vectors, and then at query time to embed the unstructured query and retrieve the embedding vectors that are 'most similar' to the embedded query. A vector store takes care of storing embedded data and performing vector search for you. For such a vector store, you can use professional (in this case open-source) products that make your life easier. The *LangChain* package provides to integrate different open-source vector databases as *Chroma*, *Pinecone FAISS* or *Lance*. Here we use FAISS (Facebook AI Similarity Search) from ai.meta (ai.meta_1 2024). FAISS is a library that allows developers to quickly search for embeddings of multimedia documents that are similar to each other. It solves limitations of traditional query search engines (SQL) that are optimized for hash-based searches, and provides more scalable similarity search functions. It includes nearest-neighbor search implementations for million-to-billion-scale datasets that optimize the memory-speed-accuracy tradeoff. FAISS aims to offer state-of-the-art performance for all operating points.

To embed text as vectors in a vector space, so that semantically similar sentences are close to each other in the vector space, a language model is required. I tested different models (HuggingFace_2 2024) to perform the embedding. In the end I decided in favour of the model *bi-encoder_msmarco_bert-base_german*, but without having made a detailed evaluation of the performance of other models.

The compilation of the vector database is then very simple. One just has to paste the text chunks to be performed and the model to be used:

```
db = FAISS.from_documents(
            chunked_documents,
            HuggingFaceEmbeddings(
            model_name='PM-AI/bi-encoder_msmarco_bert-base_german'
            )
            )
```

4.5 Creating a retriever

The `lanagchain.vectorstores.faiss` package provides a methode `as_retriever` to use the vector database as a retriever. With this method you pass two paramters:

- `search_type` (optional, [str]) – Defines the type of search that the retriever should perform. Can be set to *similarity* (default), *mmr*, or *similarity_score_threshold*.
- `search_kwargs` (optional, [Dict]) – Keyword arguments to pass to the search function.

The parameter `search_kwargs` includes things like the the number of documents

that the retriever should return based on the question. Normally this is more than just one text chunk. Here, the number of document chunks to be selected is set to k = 4.

There are potentially even more settings in this parameter. For example, `score_treshold` can be used to specify a minimum significance threshold for the similarity, below which no document is returned. `fetch_k` controls the number of documents that are included in the retriever algorithm `lambda_mult` (a number between 0 and 1) controls the diversity of responses and finally the 'filter' argument allows the search to be restricted to specific documents, whereby the filter refers to the metadata of the documents (ai.meta_2 2024).

```
retriever = db.as_retriever(
            search_type="similarity",
            search_kwargs={'k': 4}
            )
```

4.6 Load and setup the LLM

In the next step, we set up the large langage model to be used for the generation part. There are various elaborate ways of using a language model for text generation within the RAG architecture. Firstly, the language model can be retrained from scratch, secondly, a pre-trained model can be fine-tuned to your own requirements or thirdly, the model can be used directly in the existing state. The large language models today are so good and so generally applicable that training or fine-tuning them yourself would only be worthwhile if, firstly, you had a lot of computing capacity and, secondly, a large amount of your own training data available. Neither of these is the case here, which is why I decided to use an existing model unchanged (frozen model).

In this case, I could also have directly selected a model that provides an API interface. However, to be able to work more independently, I decided to download the model directly from *HuggingFace*. I chose the model *Mixtral-8x7B-v0.1* (HuggingFace_3 2024). *Mixtral* is a sparse mixture-of-experts network. It is a decoder-only model where the feedforward block picks from a set of 8 distinct groups of parameters. At every layer, for every token, a router network chooses two of these groups (the "experts") to process the token and combine their output additively. This technique increases the number of parameters of a model while controlling cost and latency, as the model only uses a fraction of the total set of parameters per token. Concretely, *Mixtral* has 46.7B total parameters but only uses 12.9B parameters per token. It, therefore, processes input and generates output at the same speed and for the same cost as a 12.9B model. *Mixtral* is pre-trained on data extracted from the open Web – experts and routers are trained simultaneously. More details about the model is provided on the website of mistral-ai (MistralAI-team 2023).

Even using a large language model without training or fine-tuning it yourself still requires a lot of memory. One way to save further computing power is to *quantise*

the model. An AI model originally uses 32-bit floating point numbers for its weights. Quantisation converts these weights into 8-bit integers. The resulting model requires less memory and can be executed faster on hardware that is more efficient with lower precision. For our purposes, however, this precision is sufficient and allows the model to be loaded and run on *GoogleColab*.

The code for quantisation is printed below. It essentially consists of setting parameters and configuring the quantisation.

```
###############################################################
# bitsandbytes parameters
###############################################################

# Activate 4-bit precision base model loading
use_4bit = True

# Compute dtype for 4-bit base models
bnb_4bit_compute_dtype = "float16"

# Quantization type (fp4 or nf4)
bnb_4bit_quant_type = "nf4"

# Activate nested quantization for 4-bit base models
# (double quantization)
use_nested_quant = False

###############################################################
# Set up quantization config
###############################################################
compute_dtype = getattr(torch, bnb_4bit_compute_dtype)

bnb_config = BitsAndBytesConfig(
            load_in_4bit=use_4bit,
            bnb_4bit_quant_type=bnb_4bit_quant_type,
            bnb_4bit_compute_dtype=compute_dtype,
            bnb_4bit_use_double_quant=use_nested_quant,
)

# Check GPU compatibility with bfloat16

if compute_dtype == torch.float16 and use_4bit:
    major, _ = torch.cuda.get_device_capability()
    if major >= 8:
        print("=" * 80)
        print("Your GPU supports bfloat16: accelerate \
            training with bf16=True")
```

```
        print("=" * 80)
```

Then I can load the model:

```
################################################################
# Load pre-trained config
################################################################
model_name= 'mistralai/Mixtral-8x7B-v0.1'

model = AutoModelForCausalLM.from_pretrained(
            model_name,
            quantization_config=bnb_config
            )
```

Finally, I also configure the tokeniser, which is of course set up with the same model.

```
################################################################
# Tokenizer
################################################################

tokenizer = AutoTokenizer.from_pretrained(
            model_name,
            trust_remote_code=True
            )

tokenizer.pad_token = tokenizer.eos_token
tokenizer.padding_side = "right"
```

4.7 Creating a LLM Chain

Now that I have loaded the model, I can build a pipeline to use the model as a text generator.

I use the pre-configured model and the defined tokeniser for this. The `temperature` specifies the degree of freedom with which the model should generate a response or, in other words, how close the response should be formulated to the text provided in the prompt. `repetition_penalty` is a further restriction that is intended to prevent repetitions. Finally, I set the number of tokens to be generated to 1000.

The pipeline is then created by the `HuggingFacePipline` function.

```
text_generation_pipeline = pipeline(
            model=model,
            tokenizer=tokenizer,
            task="text-generation",
            temperature=0.2,
```

```
            repetition_penalty=1.1,
            return_full_text=True,
            do_sample = True,
            max_new_tokens=1000,
            )

mistral_llm = HuggingFacePipeline(
            pipeline=text_generation_pipeline
            )
```

Now I have to create a prompt in which the original question, the result from the retriever process and further instructions are passed to the generator. `{context}` are the document chunks found by the retriever and `{question}` is the original question.

```
prompt_template = """
### [INST]

Instruction: Beantworte die Frage aufgrund deines \
    Wissens über die PHBern. Antworte in **ZWEI bis \
    DREI DEUTSCHEN Sätzen**! Hier ist der Kontext als Hilfe:

{context}

### QUESTION:
{question}

[/INST]

 """
```

And then the prompt is generated with the prompt template and the input variables `{context}` and `{question}`:

```
# Create prompt from prompt template
prompt = PromptTemplate(
            input_variables=["context", "question"],
            template=prompt_template,
            )
```

Then I use the Generator-Pipeline `mistral_llm` together with the prepared prompt `prompt` to form a chain named `llm_chain`.

```
# Create llm chain by pasting together the
# Text_generation-Pipeline 'mistral_lmm' and the prompt
llm_chain = LLMChain(llm=mistral_llm, prompt=prompt)
```

4.8 Pasting the pipelines together to a RAG Chain

The RAG Chain `rag_chain` in the next piece of code is the all glued together: the generator chain `llm_chain` (in whitch the generator pipeline `mistral_llm` and the prompt template `prompt` is pasted together), augmented by the `question` and by the external `context`, found by the retriever.

```
rag_chain = (
        {
        "context": retriever,
        "question": RunnablePassthrough()
        } | llm_chain
        )
```

4.9 First Tests - First Results

Now that I have set up my RAG chain, I can put it through its first tests. Let's assume we have the following question:

"Wie lange werden meine Arbeitszeugnisse an der PHBern aufbewahrt?"

If we ask ChatGPT 4o we get the following response:

"An der PHBern werden Arbeitszeugnisse gemäß den gesetzlichen Vorgaben des Schweizerischen Obligationenrechts (OR) und den internen Richtlinien der Hochschule aufbewahrt. In der Regel beträgt die gesetzliche Aufbewahrungsfrist für Arbeitszeugnisse 10 Jahre. Diese Frist beginnt ab dem Ende des Arbeitsverhältnisses.

Es empfiehlt sich jedoch, bei der Personalabteilung der PHBern nach den spezifischen internen Richtlinien zu fragen, da die Hochschule möglicherweise eigene Aufbewahrungsfristen hat oder zusätzliche Regelungen beachtet. Die Personalabteilung kann dir genauere Informationen über die Aufbewahrungsdauer und die Modalitäten der Archivierung deiner Arbeitszeugnisse geben."

ChatGPT is not able to give precise information about PHBern internal regulations and answers in general terms. We will now try, whether our RAG can do better.

The method `invoke` applies our RAG chain. The question the RAG mechanism should answer is expected as input. So we put our query into the function:

```
result = rag_chain.invoke(
"Wie lange werden meine Arbeitszeugnisse an der PHBern aufbewahrt?"
)
```

We assign the execution of the RAG chain to a variable `result` so that we can access specific aspects of the rag_chain result. For example, we can use `result['context']` to see which text chunks are passed to the generator for text generation.

```
result['context']
```

The output is a list of *LangChain* documents with the content the retriever has found in the original texts, with each document containing a tuple of entries; the `page_content` and the `metadata`. The `page_content` is the relevant text as a string and the `metadata`is a dictionary with `document`as the *key* and the title of the document the text originates from as the *value*. As it was setup in its parameters, the list contains the 4 most relevant text chunks of 500 characters each. It looks like this:

```
[Document(
    page_content=
    'erenden des Abschlusskolloquiums, spätestens jedoch \
    zwei Wochen vor dem Abschlusskolloquium. Die \
    Zeitfenster sind im Anlassverzeichnis aufgeführt. Die \
    Studierenden arbeiten während der gesamten Ausbildung \
    am IS2 der PHBern am Berufskonzept. Sie schreiben \
    sich in die Lerngelegenheiten des Moduls Transfer \
    und Vernetzung parallel zur Einschreibung in die \
    Module Einführungspraktikum und/oder Fachpraktikum \
    wie folgt ein: Lerngelegenheit, Voraussetzung, \
    Einschreibung, Orientierungstag. Es erfolgt auto',

    metadata={'document': 'Wegleitung TRANSFER UND VERNETZUNG.'}
    ),

 Document(
    page_content='Informationen des Rechtsdiensts zur \
    Entschädigung bei Praktika.\n Die Studierenden der \
    PHBern absolvieren in allen Grundausbildungsstudiengängen \
    eine Berufspraktische Ausbildung. Diese besteht aus \
    „praktisch angelegten, professionell begleiteten Lehr- \
    und Lernanlässen, welche im unmittelbaren Kontakt mit \
    dem Berufsfeld der Förderung der Handlungskompetenz \
    als Lehrperson dienen", und damit primär aus Praktika. \
    Der Begriff „Praktikum" hat in der Lehrerinnen- und \
    Lehrerausbildung eine lange Tradition',

    metadata={'document': 'Informationen des Rechtsdiensts \
    zur Entschädigung bei Praktika.'}
    ),

 Document(
    page_content='Akten und Dateien, die Personendaten \
    enthalten, sind grundsätzlich zu vernichten, \
    sobald sie nicht mehr benötigt werden. Personaldaten \
```

```
    sind nach erfolgtem Stellenaustritt während fünf \
    Jahren aufzubewahren; anschliessend werden sie \
    vernichtet. Arbeitszeugnisse und Bestätigungen \
    sowie für deren Ausstellung notwendige Unterlagen \
    werden zehn Jahre nach erfolgtem Stellenaustritt \
    vernichtet. Unterlagen über von Studierenden \
    erbrachte Leistungsnachweise, insbesondere \
    Prüfungsarbeiten, Prüfungsprotokol',

    metadata={'document': 'Weisungen über den Umgang \
    mit Personendaten (Datenschutzweisungen) vom 29. Juni 2021.'}
    ),

 Document(
    page_content='Der Studienplan kann \
    Teilnahmevoraussetzungen für einzelne \
    Lehrveranstaltungen festlegen. Die Fakultät und \
    die PHBern sorgen dafür, dass die entsprechenden \
    Lehrveranstaltungen im durch den Studienplan \
    festgelegten Turnus angeboten werden. Die \
    Regelstudienzeit beträgt vier Semester. Wer \
    ohne wichtigen Grund (Art. 35 Abs. 1 der \
    Verordnung vom 12. September 2012 über die \
    Universität (UniV)) länger als acht Semester \
    studiert, wird vom Weiterstudium ausgeschlossen. \
    Die Bewilligung für eine Verlängeru',

    metadata={'document': 'Reglement für den \
    spezialisierten Joint Master-Studiengang \
    Fachdidaktik Sport der Philosophisch-\
    humanwissenschaftlichen Fakultät der \
    Universität Bern und der Pädagogischen \
    Hochschule Bern (RFd Sport) vom 18. Mai \
    2015 und 16. Juni 2015 (Stand am 1. Februar 2022).'}
    )
]
```

To get the generated answer we have to print `result['text']`.

```
print(result['context'])
```

And we get a reasonable answer!

```
[
'Die Arbeitszeugnisse werden zehn Jahre nach dem \
Stellenaustritt vernichtet. (Zehn Jahre nach dem \
Ausscheiden aus der PHBern werden die Arbeitszeugnisse \
```

```
aufbewahrt und anschließend vernichtet.)'
]
```

In the prompt, the generator was instructed to answer in '*zwei bis drei Deutschen Sätzen*'. This is probably the reason why the same answer is formulated a second time in redundant form in brackets.

But the result is cause for optimism. The RAG works and produces a reasonable result. Whether this is really correct is another question and would have to be checked separately.

5 Results

As described in the previous chapter, the first tests of the self-built RAG were successful. In order to analyse the results of the RAG more systematically, the idea was now to ask the system a whole selection of questions containing content from all the documents available to the RAG.

However, in order to obtain a statement about the quality of the answers, they would have to be validated in some way. In a first step, this was done 'by hand'. I have answered some questions myself, also on the basis of the information in the documents. These answers form the reference basis (ground truth, GT). I had the same questions answered by the RAG and compared them with the ground truth. Some examples you find in the Appendix 2 in Chapter 9.2.

Different types of results can be identified:

- The predicted answer (P) is very similar to the ground truth (GT) (i.e. example 1 and 3, Chapters 9.2.1, 9.2.3)
- P is even more precise than the GT (i.e. example 2, 5 and 6, Chapters 9.2.2, 9.2.5, 9.2.6)
- P is much more detailed than the GT and answers more than was asked (i.e. example 4, Chapter 9.2.4)
- P is given in english although the prompt explicitely asks to give the answer in german (i.e. example 7, Chapter 9.2.7)
- P is grammatically correct but simply wrong in terms of content (i.e. example 8, Chapter 9.2.8)

5.1 Extended RAG

As an extension of the RAG, an additional question loop can be added in the next step. This requires the opening of a history, which is added to the prompt template. The challenge is to deal with follow-up questions in which reference is made to the previous question as in this example:

question1: "*Wie lange dauert ein Studium für das Lehrdiplom für Maturitätsschulen?*"

question2: "*Welche Voraussetzungen brauche ich dazu?*"

In order to cope with this, the prompt needs to be elaborately designed. In the prompt, the generator must be given an example of how it should react if a history is present or not.

```
_template = """
[INST]
Given the following conversation and a follow up question,
rephrase the follow up question to be a standalone question,
IN ITS ORIGINAL LANGUAGE, which is GERMAN, that can be used
to query a FAISS index. This query will be used to retrieve
documents with additional context.

Let me share a couple examples that will be important.

If you do not see any chat history, you MUST return the
"Follow Up Input" as is:

'''
Chat History:


Follow Up Input:
Wieviele ECTS Punkte muss ich im Studiengang Schulische
Heilpädagogik studieren?

Standalone Question:
Wieviele ECTS Punkte muss ich im Studiengang Schulische
Heilpädagogik studieren?
'''

If this is the second question onwards, you should
properly rephrase the question like this:

'''
Chat History:
Human: Wieviele ECTS Punkte muss ich im Studiengang
Schulische Heilpädagogik studieren?

AI:
Im Studiengang Schulische Heilpädagogik werden
60 ECTS Punkte absolviert.

Follow Up Input:
Und welche Voraussetzungen sind dafür nötig?
```

```
Standalone Question:
Welche Voraussetzungen sind für den Studiengang
Schulische Heilpädagogik nötig?
'''

Now, with those examples, here is the actual chat
history and input question.

Chat History:
{chat_history}

Follow Up Input:
{question}

Standalone question:
[your response here]
[/INST]
"""

CONDENSE_QUESTION_PROMPT = PromptTemplate.from_template(_template)
```

The complete code for this extended *Chat-RAG* can be found in (Thaker 2024). I have adapted and used this template for my purposes and it works satisfactorily.

5.2 Benchmarking

This first qualitative inspection leads to the question of whether the comparison between the predictions and the ground truth can be quantified. In fact, there are various metrics to measure semantic similarity between words, sentences or texts. A few are described here.

5.2.1 Metrics

CosineSimilarity (Wikipedia 2024):

The *CosineSimilarity* is a measure of the similarity between two vectors in the semantic manyfold vector space. The cosine of the angle between the two such vectors is determined. The cosine of the included zero angle is one, meaning total semantic similarity; for every other angle, the cosine of the included angle is a number between zero and one. An angle of zero means total independence of two vectors or, in this context, total semantic independence.

BLEU (Sakib Haque 2022):

BLEU, or *Bilingual Evaluation Understudy* is a precision based measure of n-gram overlap. It compares n-grams in the prediction and reference outputs. An average *BLEU* score is computed by combining *BLEU1* to *BLEU_n* using

predetermined weights for each n. For our evaluation, we report *BLEU1* and the average *BLEU* score with n ranging from 1 to 4 and weights 0.25 for each n.

ROUGE (Lin 2004):

ROUGE, or *Recall-Oriented Understudy for Gisting Evaluation*, is a set of metrics and a software package used for evaluating automatic summarization and machine translation software in natural language processing. The metrics compare an automatically produced summary or translation against a reference or a set of references (human-produced) summary or translation. ROUGE metrics range between 0 and 1, with higher scores indicating higher similarity between the automatically produced summary and the reference.

The following five evaluation metrics are available.

- *ROUGE-N*: Overlap of n-grams between the system and reference summaries.
- *ROUGE-1*: refers to the overlap of unigrams (each word) between the system and reference summaries.
- *ROUGE-2*: refers to the overlap of bigrams between the system and reference summaries.
- *ROUGE-L*: Longest Common Subsequence (LCS) based statistics. Longest common subsequence problem takes into account sentence-level structure similarity naturally and identifies longest co-occurring in sequence n-grams automatically.
- *ROUGE-W*: Weighted LCS-based statistics that favors consecutive LCSes.
- *ROUGE-S*: Skip-bigram based co-occurrence statistics. Skip-bigram is any pair of words in their sentence order.
- *ROUGE-SU*: Skip-bigram plus unigram-based co-occurrence statistics.

BertScore (Sakib Haque 2022):

BertScore measures sentence similarity using *BERT* based embeddings. *BertScore* represents each sentence as a sequence of tokens where each token is a word in the sentence. It then uses a pre-trained contextual embedding of different variants of *BERT*. Next it computes the pairwise inner product (pre-normalized cosine similarity) between every token in the reference and predicted output. Finally it matches every token in the reference and predicted output to compute the *precision*, *recall* and *F1* measure.

There are many other, more specific and sophisticated metrics (Sakib Haque 2022) that I can't go into here. However, some of them are listed here:

InferSent uses a siamese neural network architecture to produce sentence embeddings.

Universal sentence encoder encodes text into high dimensional vectors using transformers.

SentenceBert is another sentence embedding technique that uses a siamese network architecture to generate fixed length representation of sentences.

Attendgru is an encoder-decoder based neural machine translation model.

5.2.2 Ground Truth

To apply any of these metrics, we need a reference database to compare the results of the RAG engine with, which forms the so-called "*ground truth (GT)*".

I have created such a database by hand. It consists of an Excel list (fig. 7) with 1033 question - answer - pairs. In detail it has the following content:

ID no.: Contains a consecutive identification number

Title: Contains the title of the document with the content to which the question refers

Question: Contains a self-formulated question

Answer: Contains a self-formulated answer that serves as a reference for assessing the result of the RAG.

Context: Contains the excerpt from the entire document on which the answer is based.

Document: Contains the entire text of the corresponding document

Figure 7: Ground Thruth Database

Each of these 1033 questions was then fed into the RAG algorithm to generate an answer. The questions, the ground-truth answers and the results (generated responses) were then logged together in a separate Excel list.

This Excel list serves as the basis for further analyses.

5.2.3 Results Benchmarking

Two metrics were used and compared for benchmarking and applied on all 1033 answer-pairs (GT-answer - RAG-answer), *CosineSimilarity* and *BertScore*. *BertScore* provides three different sub-metrics; *BertScore_Precission* *BertScore_Recall* and *BertScore_F1*.

Both metrics require a model that can be used to tokanise and build the semantic vector space. The benchmarking was tested with three different *BERT*-models:

- *distilbert-base-uncased* (HuggingFace_4 2024)
- *bert-base-multilingual-cased* (HuggingFace_5 2024)
- *xlm-roberta-base* (HuggingFace_6 2024)

To get an idea of the difference between the various metrics, the four metrics *CosineSimilarity*, *BertScore_Precission*, *BertScore_Recall* and *BertScore_F1* were each calculated and compared using the same model. The *distilbert-base-uncased* model was used for this purpose. The result can be seen in Figure 8. It shows the distribution (histogram) of the scores of the various metrics applied to the 1033 results of the RAG.

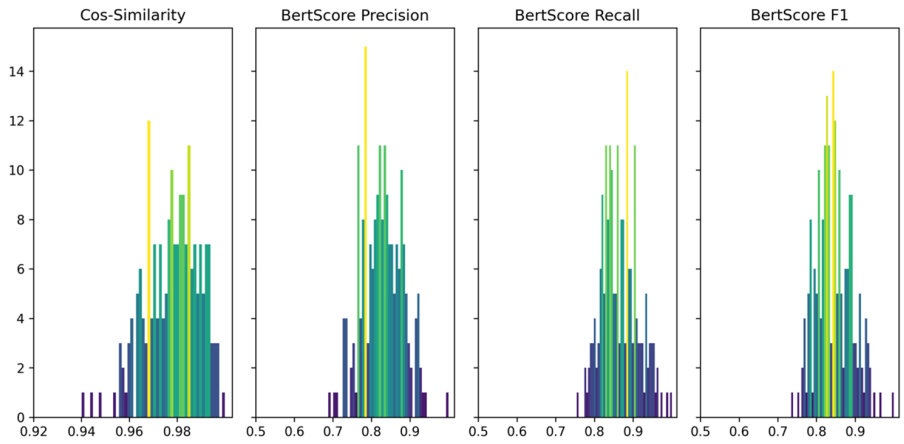

Figure 8: Histogram for the scores of the four metrics applied on the RAG results

The structure of the distribution of scores for the various metrics is similar. However, the range of scores for *CosineSimilarity* is between 0.94 and 1.0, while the *BertScore* metrics are between approx. 0.7 and 1.0.

Since the *BertScore_F1* is the harmonic mean of *BertScore_Precission* and *BertScore_Recall* (equation 1), I decided to analyse the different models using only the *BertScore_F1*.

$$F_1 = \frac{2}{\text{precision}^{-1} \cdot \text{recall}^{-1}} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \tag{1}$$

So I tried to find out whether there are differences in the determination of the *BertScore_F1* depending on the model used. As described above, I compared three *BERT* models calculating the *BertScore_F1*. The results of this comparison can be seen in Figure 9.

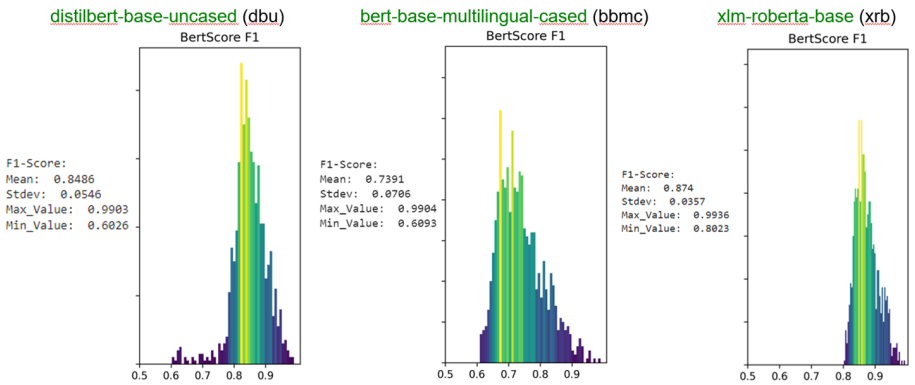

Figure 9: Histogram for the *BertScore_F1* calculated with three different *BERT* models

In order to evaluate which model is the most suitable for calculating the BertScore, it is necessary to determine which properties are expected and what the benchmarking should achieve.

Surely benchmarking should enable a good discrimination between good and bad quality answers. The BertScore should therefore have as wide a spread as possible. In order to quantify this for the three test results, the most important statistical values are printed in Figure 9. They can also be seen in Table 1.

Model	Mean	Stdev	Max_Val	Min_Val	Δ-Val
distilbert-base-uncased	0.8486	0.0546	0.9903	0.6026	0.3877
bert-base-multilingual-cased	0.7391	0.0706	0.9904	0.6093	0.3811
xlm-roberta-base	0.8740	0.0357	0.9936	0.8023	0.1913

Table 1: Performance metrics for different models

The range of values for the *distilbert-base-uncased* model is the widest, and the distribution is such that "bad values" (downward outliers) stand out clearly from the mean. I have therefore decided to use the *BertScore_F1* with the *distilbert-base-uncased* model as the metric for benchmarking.

5.2.4 Analysing bad scored answers

When looking at the distribution of the scores (*BertScore_F1* with model *distilbert-base-uncased*), it can be seen that the frequency of the values starts to rise sharply from values around 0.78. When analysing the "bad answers", I therefore concentrate on those with a score < 0.78.

56 answers have a score lower than 0.78, which corresponds to 5.4%. 3.9% (41 answers) have a score lower than 0.76. 34 of these 41 answers have a poor score because the answers were given in English. The poor score therefore does not indicate poor semantics, but only that the instruction in the prompt was not followed. Only 7 of the 41 lowest ranked answers are in German.

Here is the worst ranked german example:

Question: *Wie gross ist der Mehraufwand, wenn ich statt einem zwei Studienschwerpunkte studiere?*

GT:
Der Zusatzaufwand beträgt auf 15 ECTS-Punkte.

Pred:
Aufgrund des bereitgestellten Dokuments kann ich nicht genau sagen, wie viel Mehraufwand entsteht, wenn man statt einem zwei Studienschwerpunkte wählt. Laut dem Dokument "Studienreglement für den Studiengang Schulische Heilpädagogik (StudR SHP)" vom 12. Juni 2018 (Stand am 1. Februar 2023) kann der Studienschwerpunkt maximal einmal gewechselt werden, was darauf hindeutet, dass normalerweise nur ein Schwerpunkt pro Studium gewählt wird. Daher ist es möglicherweise nicht möglich, zwei Studienschwerpunkte zu absolvieren, ohne zusätzlichen Aufwand zu haben. Ob und in welchem Umfang ein Mehraufwand entsteht, hängt von den spezifischen Vorgaben des Studiengangs ab.

The example shows that even answers with poor rankings in the score can be meaningful answers. The question in this example is very general and is difficult to answer due to the documents. The RAG system reveals that it has difficulties in finding an answer. Nevertheless, it then tries to provide information and becomes entangled in a contradiction (impossibility of taking two majors and at

the same time the statement that the additional effort depends on the degree programme).

In a second example, we see that despite a high score, the answer is simply wrong:

Question: *Wie lange dauert die Prüfungslektion in der Regel?*

GT:
Die Prüfungslektion dauert in der Regel 45 Minuten

Pred:
Die Dauer einer Prüfungslektion hängt davon ab, ob es sich um eine schriftliche, mündliche oder praktische Prüfung handelt. Schriftliche Prüfungen dauern in der Regel zwischen 60 und 180 Minuten, während mündliche Prüfungen pro Person zwischen 15 und 45 Minuten dauern. Praktische Prüfungen dauern ebenfalls pro Person maximal 180 Minuten.

This example is typical of some of the mistakes made by the RAG. It cannot semantically distinguish between two outwardly similar words: in this case, the terms "*Prüfung*" and "*Prüfungslektion*".

Another frequent occuring error that can clearly be attributed to the retriever part is the referencing of the answer to incorrect basic documents. The generator may then not find a precise answer, which it then discloses. This is usually due to the imprecise question, from which it is not possible to deduce which degree programme the question refers to.

5.3 Evaluation

As a preliminary conclusion, it can be said here that the *BertScore* gives an indication of a poor or good match between the word sequences of the ground truth and the generated answer, but this does not always mean that the answer is semantically incorrect or correct.

Despite these shortcomings of benchmarking with the scores described, it is the only way to obtain a quantitative measure of the quality of the results. A better analysis of the results, in particular checking the correctness of the content of the answers, leads to a more comprehensive manual analysis of the quality of the answers. This was started as part of the project but has not yet been completed.

I also tried to upload the results to *ChatGPT 4o* and get the system to analyse the results. As a first solution, *ChatGPT 4o* creates a benchmarking comparable to my analysis with the scores also used. I then tried to get *ChatGPT 4o* to

analyse and classify the quality of the answers as I would do. However, this did not lead to satisfactory results.

To get better results I also tried to do a pre-processing of the original text.

The following steps were made to simplify the original text:

- removing HTML tags
- removing URLs
- removing punctuation
- converting text to lowercase
- removing numbers
- removing stopwords
- lemmatization

Here is the function that served this purpose.

```
def clean_text(text):
    # Remove HTML tags
    text = re.sub(r'<.*?>', '', text)

    # Remove URLs
    text = re.sub(
        r'http\S+|www\S+|https\S+', '', text, flags=re \
        MULTILINE
        )

    # Remove punctuation
    text = text.translate(
        str.maketrans('', '', string.punctuation)
        )

    # Convert text to lowercase
    text = text.lower()

    # Remove numbers
    text = re.sub(r'\d+', '', text)

    # Remove stopwords
    stop_words = set(stopwords.words('german'))
    tokens = word_tokenize(text)
    filtered_tokens = [
        word for word in tokens if word not in stop_words
        ]

    # Lemmatization
    lemmatizer = WordNetLemmatizer()
    lemmatized_text = [
```

```
        lemmatizer.lemmatize(token) for token in filtered_tokens
        ]

    # Re-join words
    clean_text = ' '.join(lemmatized_text)

    return clean_text
```

It is hardly surprising that the whole process with cleaned text does not lead to pleasing results. Some of them are obvious, because numbers, for example, play an important role in the text, as some questions have to be answered with concrete numerical data. But also the omission of filler words (stopwords) or the simplification of words with lemmatisation reduces the information in the text and leads to much worse results.

The conclusion from this exercise was that it is important to leave the maximum amount of information in the system and to keep the text as original as possible.

6 Deployment

I have invested a lot of time in building a simple front-end application that makes my RAG easy to use as a client. I used *Stramlit* for this. *Streamlit* is a simple way to build a Python-based website to allow the user to interact with the model. The corresponding Python code can be integrated directly into the RAG application. When the code is executed, a web page is automatically created, via which the question to be answered can be entered. The RAG mechanism then runs in the background and the result is displayed on the website.

The problem with this simple front-end solution is that *Streamlit* cannot be started from *GoogleColab* because *Google* does not provide space as a web host. However, I have to work on *GoogleColab* because the size of my model does not allow me to work with it locally. In the end, I was unable to install a functioning front-end solution due to the hardware options available to me.

As a result, I can currently only work with the RAG in *GoogleColab* - and even then only if I can work with an A100 GPU.

7 Conlcusion and Outlook

However, the very application that I have built in this project is already being offered professionally today in outstanding quality. Users can upload a certain number of documents that serve as a library for the retriever in order to obtain domain-specific information.

By building a RAG by my own I got deep insight into RAG technology and it enabled me to create my own RAG according to my needs and thus be independent of commercial or freely accessible applications from professional providers.

My project can be improved and expanded at various points.

architektur

1. The entire RAG mechanism can be expanded and refined almost at will. A first step in this direction is the expansion of the RAG prototype into a chat RAG, whereby the RAG mechanism does not simply answer questions, but can also respond to enquiries and conduct a chat. That's what I tried and what is discribed in chapter 5.1.

model

2. In my project, the large generator language model is used in a "frozen" state. In order to better adapt the model to the content of the specific domain, the model could be fine-tuned or even trained from scratch. However, both require a lot of training data and a great deal of computing and storage capacity.

3. For both the retriever part and the generator part, one could experiment with alternative models.

4. In order to improve the quality of the answers, one would have to systematically experiment with the hyperparameters of the models used for the retriever and the gernerator.

data

5. The number of documents available to the Retriever could be increased even further.
6. You could try using "Data-Augemntation" to extract information from the text even more efficiently.

deployment

7. What I really want to do next is to make the RAG accessible to a user by building a functioning front-end application. *Streamlit* seems to be a suitable simple solution here. Managing the interfaces between the place where the model can run (e.g. *GoogleColab*), the web server on which *Streamlit* runs, the platform from which the model is obtained (*HuggingFace*) and the data storage where the text documents and the vector database are stored (*GoogleDrive*) is not easy (10). I still have to find a solution here.

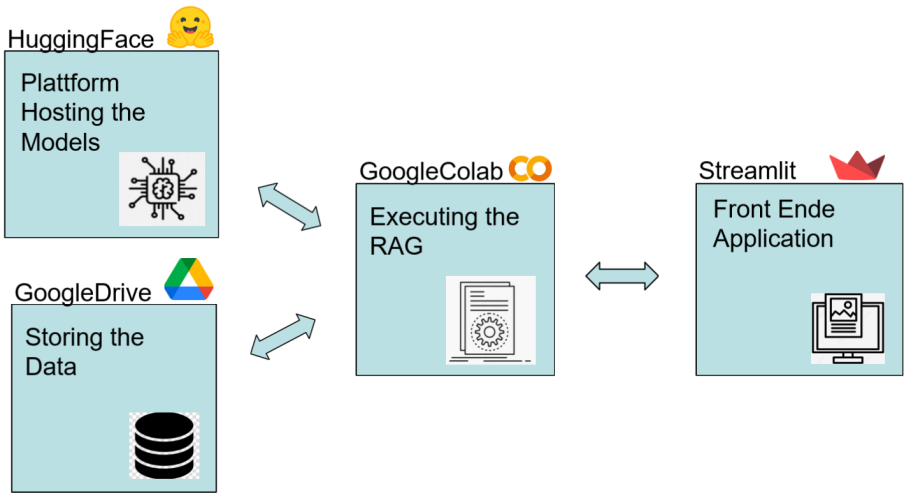

Figure 10: Interfaces between different players in the RAG App

In order to judge whether it is worthwhile for a company to operate its own RAG, it depends on the requirements for independence and flexibility. If a company wants to keep its documents and information under control and keep the option of frequently adapting and replacing the basic documents open, then it may well be worth investing in the development of its own RAG. The problem of error control is still the biggest challenge. If you want the AI to be perfect

or the error rate to be reduced as much as possible, you need additional control loops (possibly AI-generated) that operate independently of the RAG.

Chatbots will improve massively in the coming months and years and commercial products will probably soon flood the market.

8 Acknowledgements

I would like to thank the following people for their help in realising this project:

Dr Sukanya Nath, Dr Mykhailo Vladymyrov, MSc Ahmad Alhineide and PD Dr Sigve Haug for accompanying the final project, Dr Wolfgang Spahn for the technical help with all kinds of questions related to programming and developing the RAG, Gabriel and Chiara for the regular mutual exchange and finally the PHBern for the financial support of the further education in the context of which this work was created.

9 Appendix

9.1 A1: Content Corpus

1. Reglement über die Aufgaben und Befugnisse der Vizerektorin oder des Vizerektors vom 20. Juni 2023.
2. Weisungen über den Umgang mit Personendaten (Datenschutzweisungen) vom 29. Juni 2021.
3. Weisungen über die Gewährung von Nachteilsausgleichsmassnahmen (NAM-Weisungen) vom 9. März 2023.
4. Reglement über die Ergänzungsprüfung (EP-Reglement)1 vom 16. Juni 2015 (Stand am 1. August 2023).
5. Allgemeine Zulassungsweisungen vom 20. Juni 2023.
6. Studienreglement für den Bachelorstudiengang Primarstufe (StudR PS) vom 17. Januar 2023 (Stand am 1. August 2023).
7. Studienreglement für das Bachelor- und Masterstudium Sekundarstufe I (StudR S1) vom 17. August 2021 (Stand am 1. August 2023).
8. Weisungen über die Zulassung zum Masterstudium Sekundarstufe I für Personen mit einem fachwissenschaftlichen Hochschulabschluss vom 23. Mai 2023.
9. Studienreglement für das Fachdiplomstudium Sekundarstufe I (StudR S1 Fachdiplom) vom 17. August 2021.
10. Weisungen über die Ausstellung von Bescheinigungen der Diplomberechtigung am Institut Sekundarstufe II vom 24. Januar 2023.
11. Studienreglement für den Studiengang Schulische Heilpädagogik (StudR SHP) vom 12. Juni 2018 (Stand am 1. Februar 2023).
12. Reglement für den spezialisierten Joint Master-Studiengang Fachdidaktik Sport der Philosophisch-humanwissenschaftlichen Fakultät der Universität

Bern und der Pädagogischen Hochschule Bern (RFd Sport) vom 18. Mai 2015 und 16. Juni 2015 (Stand am 1. Februar 2022).

13. Studienreglement für den Masterstudiengang Fachdidaktik Textiles und Technisches Gestalten Design (StudR FD TTG-D) vom 13. Juni 2017 (Stand am 1. August 2023).
14. Studienreglement für den Masterstudiengang Fachdidaktik Natur, Mensch, Gesellschaft und Nachhaltige Entwicklung (StudR FD NMG+NE) vom 12. Juni 2018 (Stand am 1. Februar 2023).
15. Weisungen über die Durchführung von Fernprüfungen (Fernprüfungsweisungen) vom 24. Januar 2023.
16. Weisungen über die Mitteilung der Ergebnisse von Leistungsnachweisen und des Ergebnisses der Ergänzungsprüfung vom 22. August 2023.
17. Studienreglement für die Weiterbildungslehrgänge (StudR WBL) vom 14. Juni 2016 (Stand am 1. Februar 2023).
18. Weisungen über die Verwendung der Abgabe für soziale, kulturelle und sportliche Einrichtungen vom 15. August 2023.
19. Personalreglement vom 6. Dezember 2022.
20. Personalrechtsweisungen vom 20. Dezember 2022.
21. Reglement über die Rekurskommission (ReKo-Reglement) vom 12. Januar 2021 (Stand am 1. August 2023).
22. Gesetz über die deutschsprachige Pädagogische Hochschule (PHG) vom 08.09.2004 (Stand 01.01.2023).
23. Informationen der Rekurskommission zum Beschwerdeverfahren.
24. Informationen des Rechtsdiensts zur Entschädigung bei Praktika.
25. Praxisänderung: Erhöhung des Mindestumfangs der fachwissenschaftlichen Ausbildung für das Lehrdiplom für Maturitätsschulen.
26. Überprüfung der Anerkennungsvoraussetzungen der Lehrdiplome für Maturitätsschulen der Pädagogischen Hochschule Bern: Ergänzungsbericht.
27. Überprüfung der Anerkennungsvoraussetzungen der Lehrdiplome für Maturitätsschulen der Pädagogischen Hochschule Bern: Ergänzungsbericht TEIL II.
28. Merkblatt Anerkennung der sechsmonatigen betrieblichen Erfahrung.
29. STUDIENPLAN Sekundarstufe II.
30. Wegleitung VERTIEFUNGSMODULE.
31. Wegleitung Bereich Integration und Erweiterung.
32. Wegleitung FACHDIDAKTIK.
33. Geschäftsordnung der Institutskonferenz des Instituts Sekundarstufe II (GO IK-IS2) vom 5. Dezember 2022.
34. Wegleitung TRANSFER UND VERNETZUNG.
35. Wegleitung BERUFSPRAKTISCHE AUSBILDUNG.
36. Wegleitung ERZIEHUNGS- UND SOZIALWISSENSCHAFTEN.
37. Wegleitung MODUL BERUFSPÄDAGOGIK
38. Weisungen über die fachwissenschaftlichen Voraussetzungen der Zulassung zum Studium und zur Diplomierung am Institut Sekundarstufe II (Zulassungsweisungen S2) vom 30. Januar 2024.

9.2 A2: Examples

9.2.1 Example 1

Question: *Wann ist eine Nachteilsausgleichsmassnahme verhältnismässig?*

GT:
Eine NAM ist verhältnismässig, wenn sie sich mit dem Bildungsauftrag der Pädagogischen Hochschule vereinbaren lässt und der durch sie ausgelöste Aufwand nicht in einem Missverhältnis zum Nutzen steht, der für die Gesuchstellerin oder den Gesuchsteller aus ihr resultiert.

Pred:
Eine Nachteilsausgleichsmassnahme (NAM) ist verhältnismässig, wenn sie mit dem Bildungsauftrag der Pädagogischen Hochschule vereinbar ist und der daraus resultierende Aufwand nicht in keinem angemessenen Verhältnis zum Nutzen steht, den die Gesuchstellerin oder der Gesuchsteller daraus zieht.

9.2.2 Example 2

Question: *Welchen Umfang haben die Studienbereiche im Bachelorstudium PS?*

GT:
Der Studienbereich BPP umfasst 48 ECTS-Punkte, der Studienbereich ESW 35 bis 45 ECTS-Punkte und der Studienbereich FWFD 70 bis 80 ECTS-Punkte.

Pred:
Der Studienbereich "Berufspraktische Studien" (BPP) umfasst 48 ECTS-Punkte, der Studienbereich "Erziehungswissenschaften und Sprachdidaktik" (ESW) enthält 35 bis 45 ECTS-Punkte und der Studienbereich "Fachwissenschaften und Fachdidaktiken" (FWFD) beinhaltet 70 bis 80 ECTS-Punkte. Hinzu kommt das thematische Studienprofil mit 5 bis 7 ECTS-Punkten und der Studienbereich Bachelorarbeit mit 10 ECTS-Punkten. Insgesamt umfasst das Bachelorstudium PS 180 ECTS-Punkte.

9.2.3 Example 3

Question: *Was kostet mich ein "admission sur dossier"-Verfahren?*

GT:
Die Gebühr für die Anmeldung zum «admission sur dossier»-Verfahren beträgt 100 Franken und die Gebühr für das Verfahren 500 Franken.

Pred:
Die Gebühr für die Anmeldung zum «admission sur dossier»-Verfahren beträgt 100 Franken und die Gebühr für das Verfahren beträgt 500 Franken.

9.2.4 Example 4

Question: *Kann die maximale Studiendauer verlängert werden?*

GT:
Bei Vorliegen wichtiger Gründe kann die Institutsleiterin oder der Institutsleiter auf Gesuch hin eine Verlängerung der maximalen Studiendauer gewähren.

Pred:
Ja, die maximale Studiendauer kann in bestimmten Fällen verlängert werden. Dies gilt für verschiedene Studiengänge wie den Masterstudiengang Fachdidaktik Textiles und Technisches Gestalten Design, den Studiengang Schulische Heilpädagogik, den Bachelorstudiengang Primarstufe sowie das Bachelor- und Masterstudium Sekundarstufe I. Die möglichen Gründe für eine Verlängerung umfassen beispielsweise Erwerbstätigkeit, Schwangerschaft, Kinderbetreuung, Betreuung erkrankter Angehöriger, Krankheit oder Unfall, Militär- oder Zivildienst, auswärtige Studienaufenthalte, die nicht anrechenbar sind, oder den Erwerb zusätzlicher studienbezogener Kompetenzen. Die Verlängerung muss spätestens während des letzten Semesters der maximalen Studiendauer beantragt werden.

9.2.5 Example 5

Question: *Gibt es Vorgaben für den Umfang der schriftlichen Modularbeit?*

GT:
Handelt es sich bei der Modularbeit um eine schriftliche Arbeit, umfasst diese in der Regel ca. 15'000 (Arbeit à 2 ECTS) bis 30'000 (Arbeit à 5 ECTS) Zeichen.

Pred:
Ja, laut der Wegleitung Bereich Integration und Erweiterung beträgt der Umfang einer schriftlichen Modulararbeit in der Regel zwischen 15'000 (Arbeit à 2 ECTS) und 30'000 (Arbeit à 5 ECTS) Zeichen. Diese Angabe bezieht sich jedoch nur auf die schriftliche Arbeit und nicht auf eventuelle ergänzende Materialien.

9.2.6 Example 6

Question: *Wie gross ist die Sollarbeitszeit pro Tag?*

GT:
Für Vollzeitbeschäftigte beträgt die tägliche Sollarbeitszeit 8 Stunden und 24 Minuten.

Pred:
Laut den Personalrechtsweisungen vom 20. Dezember 2022 beträgt die tägliche Sollarbeitszeit für Vollzeitbeschäftigte 8 Stunden und 24 Minuten.

9.2.7 Example 7

Question: *Welche Inhalte werden im Vertiefungsmodul "Erweiterter Berufsauftrag von Lehrpersonen" bearbeitet?*

GT:
Neben der Kernaufgabe des Unterrichtens übernehmen Lehrpersonen an den Schulen andere Aufgaben, mit denen man sich im Modul Erweiterter Berufsauftrag von Lehrpersonen auseinandersetzt. Dazu gehören Themen wie das Klassenlehrpersonen-Amt, die Organisation von Lagern oder ausserschulischen Lernangeboten, oder die Mitarbeit in Fachgruppen und Schulentwicklungsprojekten.

Pred:
Based on the provided documents, the specific content of the "Erweiterter Berufsauftrag von Lehrpersonen" Vertiefungsmodul is not detailed. However, it can be inferred that this module focuses on preparing students for additional responsibilities and challenges they will encounter as educators, such as taking on the role of a class teacher, organizing school trips or extracurricular learning opportunities, and participating in subject groups and school development projects.

9.2.8 Example 8

Question: *Wie sieht die Notenskala an der PHBern aus?*

GT:
Benotete Leistungsnachweise werden nach folgender Notenskala bewertet: 6 ausgezeichnet; 5.5 sehr gut; 5 gut; 4.5 befriedigend; 4 ausreichend; 3 ungenügend; 2 stark ungenügend.

Pred:
Die Notenskala an der PHBern reicht von 1 (sehr gut) bis 6 (ungenügend). Notendurchschnitte unter 4 werden auf die nächste ganze Note abgerundet, während Noten oberhalb von 5 aufgerundet werden.

Bibliography

ai.meta_1. 2024. “FAISS Vector Database.” https://ai.meta.com/tools/faiss/.

ai.meta_2. 2024. “FAISS As_retriever.” https://api.python.langchain.com/en/latest/vectorstores/langchain_community.vectorstores.faiss.FAISS.html.

Gao, Yunfan. 2023. “Retrieval-Augmented Generation for Large Language Models: A Survey.” *Https://Doi.org/10.48550/arXiv.2312.10997*.

Ghosh, Bijit. 2024. “Designing High-Performing RAG Systems.” https://medium.com/@bijit211987/designing-high-performing-rag-systems-464260b76815.

HuggingFace_1. 2024. “Question Answering.” https://huggingface.co/docs/transformers/en/tasks/question_answering.

HuggingFace_2. 2024. “General.” https://huggingface.co.

HuggingFace_3. 2024. “Model Mistral 8x7B.” https://huggingface.co/mistralai/Mixtral-8x7B-v0.1.

HuggingFace_4. 2024. “Model Distilbert-Base-Uncased.” https://huggingface.co/distilbert/distilbert-base-uncased.

HuggingFace_5. 2024. “Model Bert-Base-Multilingual-Cased.” https://huggingface.co/google-bert/bert-base-multilingual-cased.

HuggingFace_6. 2024. “Model Xlm-Roberta-Base.” https://huggingface.co/FacebookAI/xlm-roberta-base.

LangChain. 2024. “Text Splitter.” https://www.langchain.com/.

Lin, Chin-Yew. 2004. “ROUGE: A Package for Automatic Evaluation of Summaries.” *In Proceedings of the Workshop on Text Summarization Branches Out (WAS 2004), Barcelona, Spain, July 25 - 26, 2004*.

MistralAI-team. 2023. “Mixtral of Experts - a High Quality Sparse Mixture-of-Experts.” https://mistral.ai/news/mixtral-of-experts/.

Niels Reimers, Iryna Gurevych. 2019. “Sentence-BERT: Sentence Embeddings Using Siamese BERT-Networks.” *Arxiv, arXiv:1908.10084, Https://Doi.org/10.48550/arXiv.1908.10084*.

Sakib Haque, Aakash Bansal, Zachary Eberhart. 2022. “Semantic Similarity Metrics for Evaluating Source Code Summarization.” *ICPC*.

Thaker, Madhav. 2024. “(Part 2) Build a Conversational RAG with Mistral-7B and LangChain.” https://medium.com/@thakermadhav/part-2-build-a-conversational-rag-with-langchain-and-mistral-7b-6a4ebe497185.

Wikipedia. 2024. “Cosine Similarity.” https://en.wikipedia.org/wiki/Cosine_similarity.