Brandenburgische
Technische Universität
Cottbus - Senftenberg

Date: 10.07.2024

- Faculty 1

- M.Sc. Artificial Intelligence

- *Student* : Nour Aldeen Dugha

- ***Matriculation number*** : 5004966

- *Mentor* : Prof. Dr. habil. Douglas Cunningham

- *Mentor in company* : Dr. Julian Daubermann

- ***Email*** : Julian.daubermann@stihl.de

- *Company* : STIHL

- Internship Report -

Title of internship **:** Internship in the field of Artificial Intelligence Conception.

Company **:** STIHL

Location **:** Waiblingen, Baden-Württemberg, Germany

Date of start **:** 01.05.2024

Type of work **:** Full time (5 days of work per week and 7 hours per day)

Number of hours completed **:** 315 hours until the date : 10.07.2024

Mentor	Signature
Dr. Julian Daubermann	

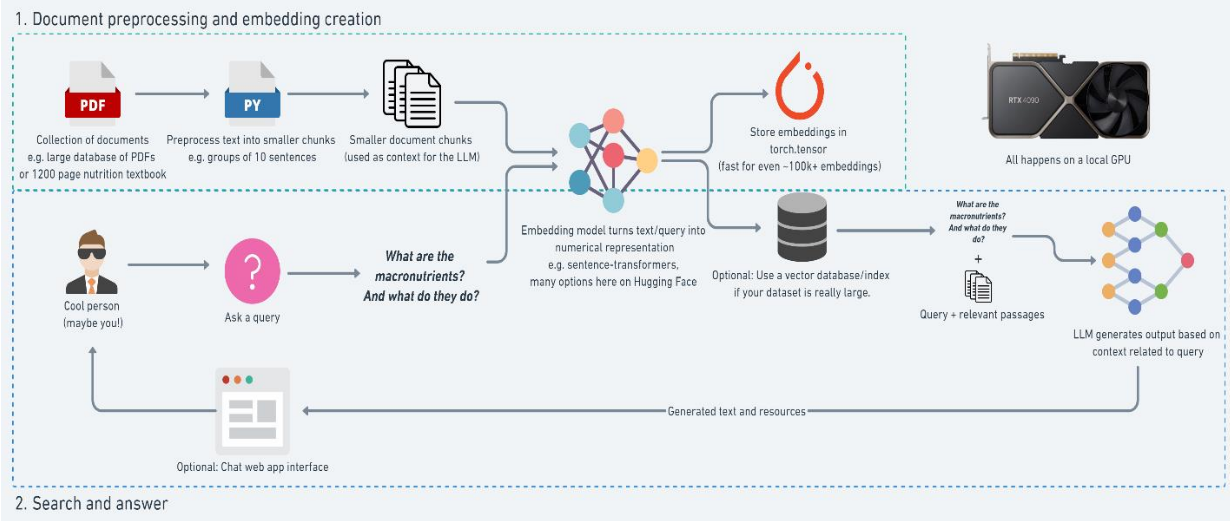
Domain of the project : the project that I am working on now in the company revolves around the Knowledge management, so basically the company has a huge amount of documents and store them as (.pdf, .doc, .docs, .txt, .xls, .csv, .ppt, .pptx, .html, and so on …) and the main aim of the project to apply the RAG (Retrieval Augmented Generation) on these documents to facilitate the retrieval process about specific thing and to leverage from the LLMs (Large Language Models) to generate a powerful response about the query or prompt that user put it to know some information or specific thing about these documents and also retrieve the most relevant top k documents that related to this query.

My tasks :

1 - Familiarize with the RAG method deeply and know the cycle of its work, and from what it is consist.

RAG: Retrieval Augmented Generation

Local RAG pipeline:


1. Document preprocessing and embedding creation
PDF
Collection of documents
e.g. large database of PDFs
or 1200 page nutrition textbook
PY
Preprocess text into smaller chunks
e.g. groups of 10 sentences
Smaller document chunks
(used as context for the LLM)
Store embeddings in
torch.tensor
(fast for even ~100k+ embeddings)
RTX 4090
All happens on a local GPU
Cool person
(maybe you!)
Ask a query
What are the
macronutrients?
And what do they do?
Embedding model turns text/query into
numerical representation
e.g. sentence-transformers,
many options here on Hugging Face
Optional: Use a vector database/index
if your dataset is really large.
What are the
macronutrients?
And what do they
do?
+
Query + relevant passages
LLM generates output based on
context related to query
Generated text and resources
Optional: Chat web app interface
2. Search and answer

What is RAG?

RAG stands for Retrieval Augmented Generation.

Each steps can be roughly broken down to:

1- **Retrieval**: seeking relevant information from a source given a query. For example, getting relevant passages of Wikipedia text from a database given a question**.**

2- **Augmented:** Using the relevant retrieved information to modify an input to a generative model (e.g. an LLM).

3- **Generating:** an output given an input. For example, in the case of an LLM, generating a passage of text given an input prompt.

Why RAG ?

The main goal of RAG is to improve the generation outputs of LLMs.

Two primary improvements can be seen as :

1- **Preventing hallucinations:** LLMs are incredible but they are prone to potential hallucination, as in, generating something that looks correct but isn't. RAG pipelines can help LLMs generate more factual outputs by providing them with factual (retrieved) inputs. And even if the generated answer from a RAG pipeline doesn't seem correct, because of retrieval, you also have access to the sources where it came from.

2- **Work with custom data:** Many base LLMs are trained with internet-scale text data. This means they have a great ability to model language, however, they often lack specific knowledge. RAG systems can provide LLMs with domain-specific data such as medical information or company documentation and thus customized their outputs to suit specific use cases.

3- **RAG** can also be a much quicker solution to implement than fine-tuning an LLM on specific data.

What kind of problems can RAG be used for ?

RAG can help anywhere there is a specific set of information that an LLM may not have in its training data (e.g. anything not publicly accessible on the internet).

For example you could use RAG for:

1- **Customer support Q&A chat:** By treating your existing customer support documentation as a resource, when a customer asks a question, you could have a system retrieve relevant documentation snippets and then have an LLM craft those snippets into an answer. Think of this as a "chatbot for your documentation". Klarna, a large financial company, uses a system like this to save $40M per year on customer support costs.

2- **Email chain analysis:** Let's say you are an insurance company with long threads of emails between customers and insurance agents. Instead of searching through each individual email, you could retrieve relevant passages and have an LLM create structured outputs of insurance claims.

3- **Company internal documentation chat:** If you have worked at a large company, you know how hard it can be to get an answer sometimes. Why not let a RAG system index your company information and have an LLM answer questions you may have? The benefit of RAG is that you will have references to resources to learn more if the LLM answer does not suffice.

4- **Textbook Q&A:** Let's say you're studying for your exams and constantly flicking through a large textbook looking for answers to your questions. RAG can help provide answers as well as references to learn more.

All of these have the common theme of retrieving relevant resources and then presenting them in an understandable way using an LLM.

Key Terms:

Token: A sub-word piece of text. For example, "hello, world!" could be split into ["hello", ",", "world", "!"]. A token can be a whole word,

part of a word or group of punctuation characters. 1 token ~= 4 characters in English, 100 tokens ~= 75 words.

Text gets broken into tokens before being passed to an LLM.

Embedding: A learned numerical representation of a piece of data. For example, a sentence of text could be represented by a vector with 768 values. Similar pieces of text (in meaning) will ideally have similar values.

Embedding model: A model designed to accept input data and output a numerical representation. For example, a text embedding model may take in 384 tokens of text and turn it into a vector of size 768. An embedding model can and often is different to an LLM model.

Similarity search/vector search: Similarity search/vector search aims to find two vectors which are close together in high-dimensional space. For example, two pieces of similar text passed through an embedding model should have a high similarity score, whereas two pieces of text about different topics will have a lower similarity score. Common similarity score measures are dot product and cosine similarity.

Large Language Model (LLM): A model which has been trained to numerically represent the patterns in text. A generative LLM will continue a sequence when given a sequence. For example, given a sequence of the text "hello, world!", a generative LLM may produce "we're going to build a RAG pipeline today!". This generation will be highly dependent on the training data and prompt.

LLM context window: The number of tokens a LLM can accept as input. For example, as of March 2024, GPT-4 has a default context window of 32k tokens (about 96 pages of text) but can go up to 128k if needed. A recent open-source LLM from Google, Gemma (March 2024) has a context window of 8,192 tokens (about 24 pages of text). A higher context window means an LLM can accept more relevant information to assist with a query. For example, in a RAG pipeline, if a model has a larger context window, it can accept more reference items from the retrieval system to aid with its generation.

Prompt: A common term for describing the input to a generative LLM. The idea of "prompt engineering" is to structure a text-based (or potentially image-based as well) input to a generative LLM in a specific way so that the generated output is ideal. This technique is possible because of a LLMs capacity for in-context learning, as in, it is able to use its representation of language to breakdown the prompt and recognize what a suitable output may be (note: the output of LLMs is probable, so terms like "may output" are used) A common term for describing the input to a generative LLM. The idea of "prompt engineering" is to structure a text-based (or potentially image-based as well) input to a generative LLM in a specific way so that the generated output is ideal. This technique is possible because of a LLMs capacity for in-context learning, as in, it is able to use its representation of language to breakdown the prompt and recognize what a suitable output may be (note: the output of LLMs is probable, so terms like "may output" are used).

Steps for building the RAG system:

We will write the code to:

1- Open our resources (documents) for example the pdfs documents.

2- Format the text of the PDF textbook ready for an embedding model (this process known as text splitting/chunking).

3- Embed all of the chunks of text in the textbook and turn them into numerical representation which we can store for later.

4- Build a retrieval system that uses vector search to find relevant chunks of text based on a query.

5- Create a prompt that incorporates the retrieved pieces of text.

6- Generate an answer to a query based on passages from the textbook.

Note: A simple method is found helpful is to break the text into **chunks** of sentences. As in, chunk a page of text into groups of 5, 7, 10 or more sentences (these values are not set in stone and can be explored).

Why do we do chunking?

1- Easier to manage similar sized chunks of text.
2- Do not overload the embedding models capacity for tokens (e.g. if an embedding model has a capacity of 384 tokens, there could be information loss if you try to embed a sequence of 400+ tokens).
3- Our LLM context window (the amount of tokens an LLM can take in) may be limited and requires compute power so we want to make sure we're using it as well as possible.

Then we would like to embed each chunk of sentences into its own numerical representation.

For example: the embedding has a shape of (768,) meaning it's a vector of 768 numbers which represent our text in high-dimensional space, too many for a human to comprehend but machines love high-dimensional space.

Note: as the information that exist in the Photo about the RAG method, actually it separates to two parts:

1- Document Ingestion and Embedding Creation.
2- RAG - Search and Answer

So now in the part search and answer:

Similarity search: Similarity search or semantic search or vector search is the idea of searching on vibe.

Whereas with similarity/semantic search, you may want to search "macronutrients functions". And get back results that don't necessarily contain the words "macronutrients functions" but get back pieces of text that match that meaning.

We can do so with the following steps:

1- Define a query string (e.g. "macronutrients functions") - note: this could be anything, specific or not.

2- Turn the query string in an embedding with same model we used to embed our text chunks.

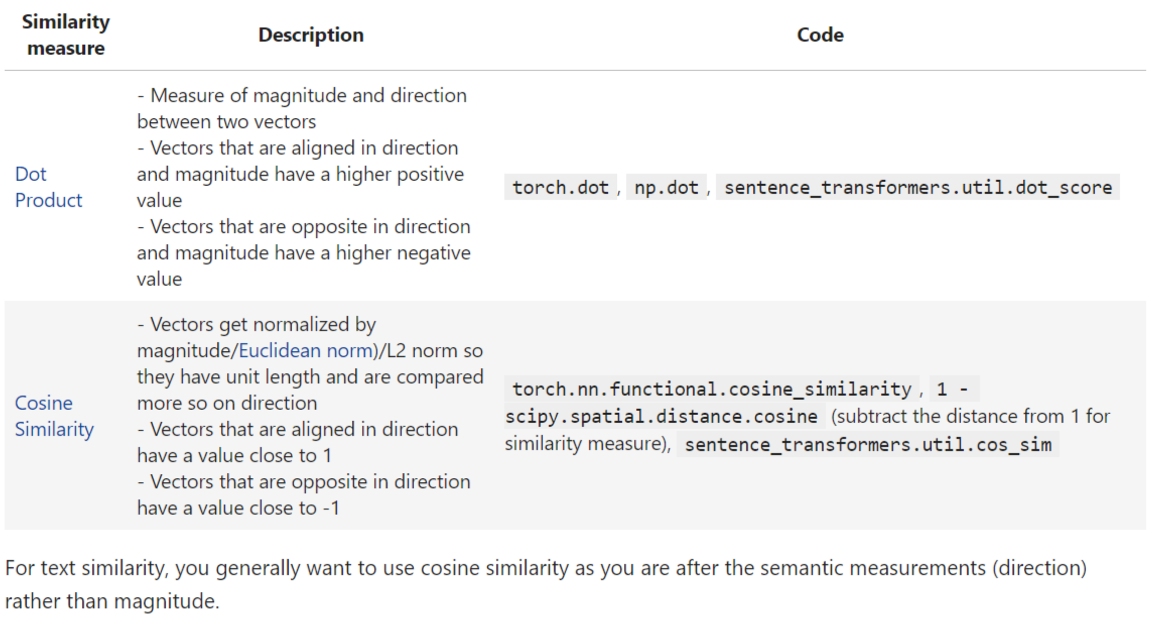
3- Perform a **dot product** or **cosine similarity** function between the text embeddings and the query embedding to get similarity scores.

4- Sort the results from step 3 in descending order (a higher score means more similarity in the eyes of the model) and use these values to inspect the texts.

Similarity measures: dot product and cosine similarity Let's talk similarity measures between vectors:

Specifically, embedding vectors which are representations of data with magnitude and direction in high dimensional space (our embedding vectors have 768 dimensions).

Two of the most common we will across are the dot product and cosine similarity. They are quite similar. The main difference is that cosine similarity has a normalization step.



Similarity measure	Description	Code
Dot Product	- Measure of magnitude and direction between two vectors - Vectors that are aligned in direction and magnitude have a higher positive value - Vectors that are opposite in direction and magnitude have a higher negative value	`torch.dot`, `np.dot`, `sentence_transformers.util.dot_score`
Cosine Similarity	- Vectors get normalized by magnitude/Euclidean norm)/L2 norm so they have unit length and are compared more so on direction - Vectors that are aligned in direction have a value close to 1 - Vectors that are opposite in direction have a value close to -1	`torch.nn.functional.cosine_similarity`, `1 - scipy.spatial.distance.cosine` (subtract the distance from 1 for similarity measure), `sentence_transformers.util.cos_sim`

For text similarity, you generally want to use cosine similarity as you are after the semantic measurements (direction) rather than magnitude.

Note: Similarity measures between vectors and embeddings can be used on any kind of embeddings, not just text embeddings. For example, you could measure image embedding similarity or audio embedding similarity. Or with text and image models like CLIP, you can measure the similarity between text and image embeddings.

VectorDB : now after we perform the embedding for the chunks we have to store these vectors of embeddings in the database.

Vector Database in RAG Systems:

Overview:

A vector database is a specialized database designed to store and manage high-dimensional vector representations of data. In the context of a RAG system, this vector database stores embeddings of textual data, which are numerical representations that capture the semantic meaning of the text.

Function in RAG Systems:

Storage of Embeddings: When textual data is processed, it is converted into embeddings using techniques like word2vec, GloVe, BERT, or other transformer-based models. These embeddings are high-dimensional vectors that encapsulate the semantic content of the text. The vector database stores these embeddings, allowing for efficient retrieval and comparison.

Efficient Retrieval: The primary purpose of the vector database in a RAG system is to enable fast and accurate retrieval of relevant information based on a query. When a user poses a query, it is also transformed into an embedding.

The vector database uses similarity search algorithms (such as cosine similarity, Euclidean distance, or other nearest neighbor search methods) to find the embeddings in the database that are closest to the query embedding. This process identifies the most semantically relevant pieces of text.

Integration with RAG Workflow: In a RAG system, the retrieval component is integrated with a generative model. After the vector database retrieves the relevant text snippets based on the query, these snippets are used as context for the generative model to produce a coherent and contextually accurate response.

This integration enhances the generative model's output by grounding it in real, retrieved data, thereby improving the relevance and accuracy of the generated content.

Advantages of Using a Vector Database Scalability:

Vector databases are optimized for handling large volumes of data, making them suitable for applications with extensive textual datasets.

Performance: Advanced indexing and search algorithms enable quick retrieval of relevant data, ensuring low-latency responses even in real-time applications.

Flexibility: They can handle various types of data and embeddings, accommodating different domains and use cases.

Popular Vector Database Several vector databases are popular in the industry, each offering unique features and optimizations:

FAISS (Facebook AI Similarity Search): Developed by Facebook AI, FAISS is a library that efficiently searches in large datasets of vectors, and this is we are using in our project.

Generation: now after we retrieve the top k most relevant documents to the query, we will pass the these relevant documents to the LLM through the prompt to generate the answer.

We create a template for the prompt that will go through the LLM to get the answer.

For example:

```
template = """
You help everyone by answering questions.
Don't try to make up an answer, if you don't know, just say that you don't know.
Answer in the same language the question was asked.
Answer in a way that is easy to understand.
Do not say "Based on the information you provided, …" or "I think the answer is…". Just
answer the question directly in detail.

Context: {context}

Question: {question}
Answer:
"""
```

So here in the above template we guide the LLM model how should will get the prompt and how should answer, the context variable will replaced by the retrieve documents from the retrieve system, and the question will replaced by the query of the user.

Tasks after done from task (1) :

2 - Familiarize with the project and its folders and files and the environment, libraries, and in particular learn the **langchain** library that will build almost all the RAG system, and learn how will integrate with the **OpenAI** library, since our embedding model is : AzureOpenAIEmbeddings() and the of it : text-embedding-ada-002, and our GPT models are, GPT4, GPT3-turbo, also learn about the steamlit library which is responsible for building the GUI for the Generative AI applications.

3 - Put two buttons after generating the answer from the LLM (thumb up and thumb down) and save what the user clicked and put the action in the file (feedback.txt), and this file contain some important information like (prompt, response, feedback of the user).

4 - Applying the hybrid search in the retrieval system of the project and this done by implement the Ensemble retriever that combine both search methods (keyword search and semantic search), this is boost and enhance our Retrieve system and give us more accurate and relevant results.

5 - Update the packages and the libraries in the project to the latest versions and fix the conflicts and the bugs and update the syntax and functions and classes to be compatible with the updates in the latest version of the libraries.

6 - Now I am working in the evaluation RAG method, this task in progress and I am doing a research about how we can evaluate the RAG system, so we have to evaluate two main components in the RAG system (Retrieval part) and (Generation part), and I am developing the evaluation method by using library called (ragas) which has 4 metrics to measure how the results of the retrieval part and the LLM answers are good.