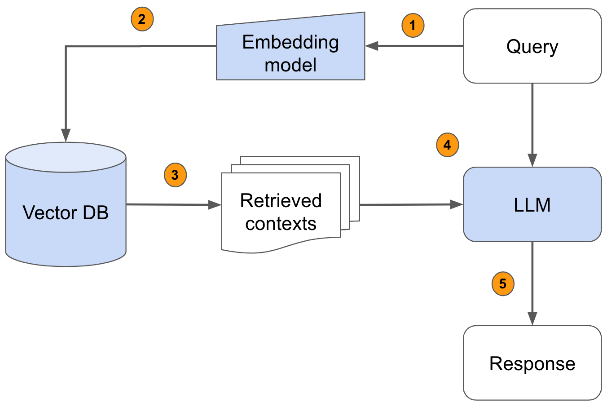
**Technical – RAG Building RAG-based LLM Applications**

For the theory about NLP & Transformers, embeddings, take a look at the other documents

# Intro

Large language models (LLMs) have limitations in terms of the information they can provide beyond their training data. Retrieval augmented generation (RAG) based LLM applications address this issue by incorporating external data sources.

**Process:**

1. passing the query through an embedding model
2. Embedding model creates an embedded query vector.This vector is then used to retrieve relevant contexts from a vector database.
3. Retrieves the top-k relevant contexts – measured by distance between the query embedding and all the embedded chunks in our knowledge base.
4. The query text and retrieved context text are then passed to a language model
5. LLM generates a response based on the provided content.

# Building the vector DB :

Let's start by studying how to store our documentation to make it usable:

First, the company's database is extracted (.pdf, .md, .txt, etc.) and divided into subparts. Each subpart can be of fixed or variable size depending on the choice of document segmentation. Overlapping techniques are possible but may introduce duplicates when selecting the appropriate chunks for answering a question.

Then, these subparts are converted into embeddings, which describe the semantic information they contain. There are many different pretrained models to choose from to embed our data but the most popular ones can be discovered through HuggingFace's Massive Text Embedding Benchmark (MTEB) leaderboard.

Finally, these embeddings are stored in a vector database.

# Retrieve the relevant chunks

We'll start by using the same embedding model we used to embed our text chunks to now embed the incoming query.

(Note : Embeddings are not the only way to determine relevant chunks of text. LLMs can also be used, but embeddings are better for retrieving the top chunks.)

From the question's embedding, a search for similar embeddings is made in the vector database. The goal of this search is to find parts of documents related to the question. We use cosine distance (<=>) but there are many options to choose from.

These contents form the context and will be added to our prompt.

# image16Generation

LLMs can then be used on these top chunks to determine the context for answering a query. Reranking or using traditional information retrieval methods like keyword matching can also be used to identify relevant chunks.

Adjusting the temperature value based on the use case is recommended. Low temperature values are suggested for factually grounded tasks, while higher temperatures can be beneficial for more creative tasks.

# Evaluation

Because we have many moving parts in our application, we need to perform both unit/component and end-to-end evaluation. Component-wise evaluation can involve evaluating our retrieval in isolation (is the best source in our set of retrieved chunks) and evaluating our LLMs response (given the best source, is the LLM able to produce a quality answer). And for end-to-end evaluation, we can assess the quality of the entire system (given the data sources, what is the quality of the response).

An RAG approach is made up of two main components:

* The vector database
* The language engine.

To evaluate our model, it is therefore necessary to study the quality of these two components, individually and together. The first component mainly depends on the choice of embedding and similarity metric. The second component depends on the choice of the language model used and the quality of the prompt.

**Using distance measures :**

There are distance measures between the response proposed by our model and the desired response. Among them, we can mention:

* Semantic distances, which study the similarity between two embeddings. They require a sentence/paragraph embedding model coupled with a similarity measure (cosine, Euclidean, etc.).
* BLEU distances (used for text translation) and ROUGE distances (used for summary evaluation), which study the similarity or overlap between two character strings based on n-grams. However, these metrics are very poorly correlated with human judgment.

**Using another LLM**

It is also possible to use another LLM to evaluate the performance of our RAG model:

* LangChain indeed offers a CriteriaEvalChain chain that allows evaluating the quality of the provided responses based on a criterion, such as conciseness or accuracy of the answer.
* The RAGAS library allows analyzing the quality of our model in several aspects (using LLMs) such as: the accuracy of the answer in relation to the context the quality of the context chosen by the retriever the ability to gather answer elements from the document database the conciseness of the provided answer.

# Best Frameworks for creating RAG applications : Langchain

## LangChain: the tool for LLM engineers.

LangChain is a framework that simplifies the implementation of chatbots based on language models. Thanks to its large Open Source community, LangChain offers many features and is now the most effective way to create architectures around language models. Among the features of LangChain, we will mention the following:

* Document Loader: allows reading texts from a source
* Vector Store: vector database for embeddings
* Retriever: tools to find documents in a DB or on the Web
* Chain: LLM processing pipeline, in several steps
* Tools: a way to connect an agent with external services - equivalent to ChatGPT plugins
* Agent: extension of chains to add tools (link with the outside of LLM)

## LlamaIndex (not tested)

Sources :

* [Building RAG-based LLM Applications for Production (anyscale.com)](https://www.anyscale.com/blog/a-comprehensive-guide-for-building-rag-based-llm-applications-part-1)
* [Construire son RAG (Retrieval Augmented Generation) grâce à langchain: L’exemple de l’Helpdesk d’OCTO - OCTO Talks !](https://blog.octo.com/le-chatbot-docto-langchain-rag-et-code-associe/)