# Case Study: Knowledge Management for Companies

Prabhal Ghosh Ishfaaq Illahibuccus Sona

**Objective of the case study**

The goal of the case study is to create a chatbot that will respond to queries from the user. More precisely, this chatbot will leverage the capabilities of a pre-trained Large Language Model (LLM) and will have as reference data the fictitious profiles of five employees of a made up Data Science company. The user will then be able to access and retrieve information from the document by asking questions to the chatbot.

**Our approach**

The data used for this case study is generated from ChatGPT 3.5 and the Large Language Model (LLM) chosen is Llama 2 (7B tokens) from Meta. The data is about the profile of five fictitious employees working at a Data Science company. The data is in the form of a PDF file and will be extracted from the file and split into a bunch of chunks.

We will then use an embedding model to convert the word in each chunk of text into numerical vectors. The embedding model (model name: sentence-transformers/all-MiniLM-L6-v2) chosen is obtained from HuggingFace and converts each word into a vector of 384 dimensions.

These embedding vectors will then be stored into a vector database. The vector database we have chosen is Pinecone, which provides a free functionality to store those vectors.

When a query is asked by the user, a similarity search will be performed between the query asked and the embedding vectors stored in the vector database. The relevant information pertaining to the query will then be retrieved from the database. We will then use the relevant information and the LLM to provide the most relevant answer to the query.

**Results**

Due to limited time and computing resources (the Llama 2 model was taking hours to respond to a single query on our laptop), we were not able to test the chatbot on several questions. However, we were able to run one query and the results obtained were very accurate. The Retrieval-Augmented Generation process was working correctly as it was able to retrieve the appropriate part of the document related to the query. The LLM was then able to use that updated context and give an answer in a correct question-answer format that we would expect from a chatbot.

```
user_input=input(f"Input Prompt:")
```

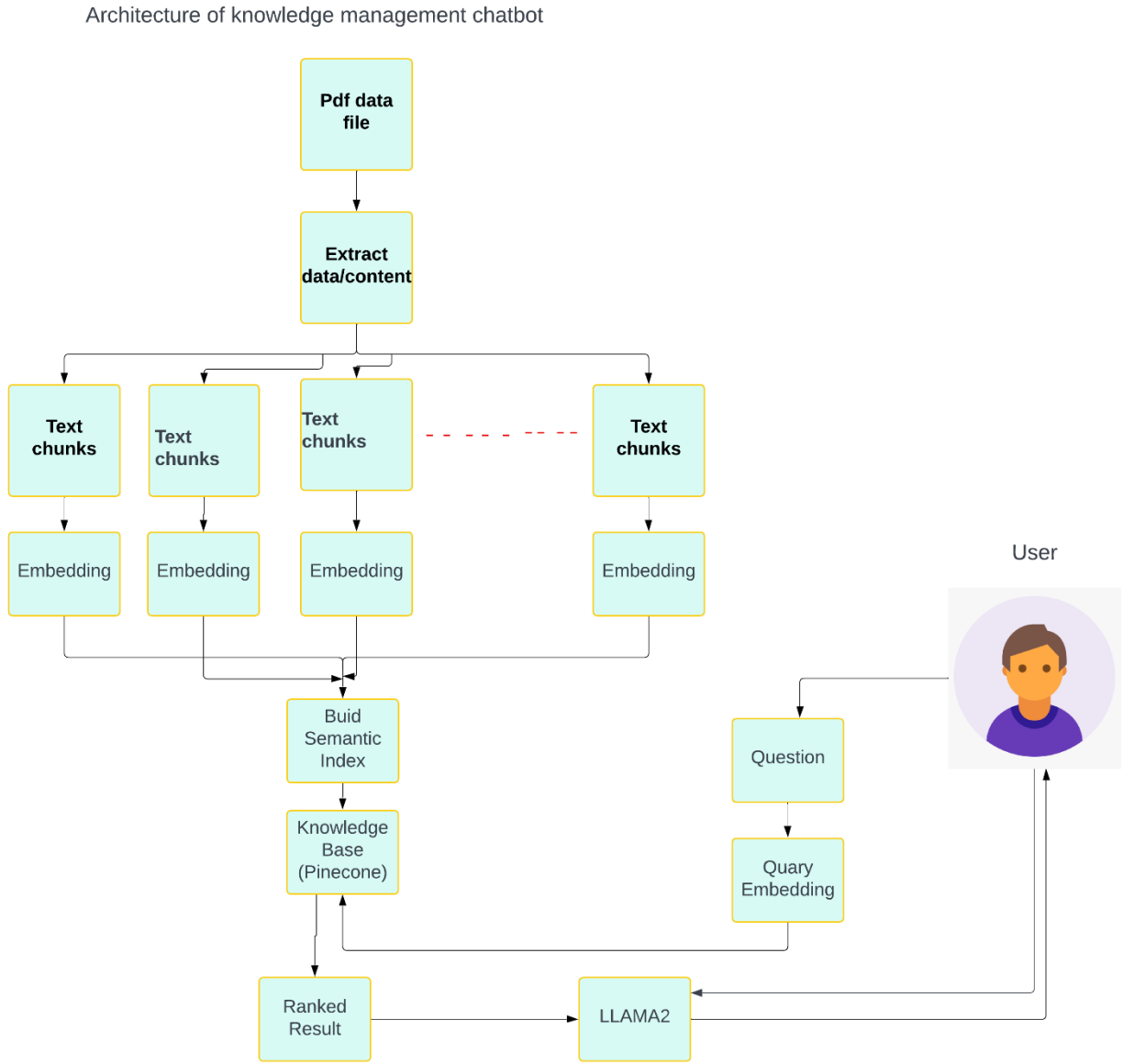
```
user_input
```

```
'Who is Alex ?'
```

```
result=question_answer({"query": user_input})
print("Response : ", result["result"])
```

```
Response :  Alex Thompson is a Machine Learning Engineer at AI Research department in the company. He joined the company on M
arch 15, 2018, and his professional background includes earning an MSc in Computer Science from Stanford University in 2017 a
nd a BEng in Electrical Engineering from MIT in 2015. His work experience includes working as a Software Engineer at Google R
esearch from 2017 to 2018 and interning at Apple AI Labs during the summer of 2016. He has skills and expertise in machine le
arning, computer science, electrical engineering, and software development.
```

## Architecture:



## Appendix

### Word Embeddings

Machine learning models and neural networks cannot work directly with text data and only take numbers or vectors of numbers as input. Hence when working with text data in the context of Natural Language Processing (NLP), the text first has to be converted to numbers (vectorization) before it is fed to the model.

One strategy to convert text to numbers is to use word embeddings. Compared to one-hot encoding which produces a very sparse representation of a word (each word is represented as an array of 1 at the position of the word zeros in all other positions and), word embeddings provide an efficient and dense representation of each word, where the dimension of the word vector is a hyperparameter to be specified by the user. The higher the dimension of the embedding chosen, more details about the relationship between words can be captured but more data will be required and the training will take longer. In addition, compared to one-hot encoding or integer encoding, word embeddings represent words as vectors in such a way that similar words will have vectors which are close to each other in the vector space. Each vector consists of floats and the values of the numbers in the vectors are learned during training, similar to the way weights are learned by a neural network (Word embeddings, 2023).

In this case study, an Embedding model from HuggingFace will be used to convert each word into a 384 dimensional vector.

### Large Language Model (LLM) – Llama 2

For the purpose of this case study, the Large Language Model (LLM) that we have chosen is Llama 2, which is a family of open-sourced LLMs released by Meta AI in 2023. According to the paper released by the Meta AI research team together with the model, Llama 2 outperforms other open-source chat models on most benchmarks and human evaluations of helpfulness and safety chosen. It also performs generally on the same level as closed-source models, with the exception of GPT-4 which comes up ahead. Most specifically for our purpose, we have chosen to use Llama 2-Chat which has been optimized for chat bots and we will use the model with the scale of 7 billion (7B) parameters (as opposed to the 13B or 70B models) due to computing power constraints (Touvron et al., 2023).

### LangChain

The framework that we have chosen to work with is LangChain, one of the most widely used language model integration frameworks in the Natural Language Processing(NLP) field currently. LangChain provides numerous tools and techniques that helps improve the accuracy of responses from language models and build new prompt templates or customize existing ones (prompt engineering).

While LLMs can generate very acceptable answers to user queries asked in a general context, they struggle to answer questions in a specific domain in which they were never trained on. LangChain allows the LLMs to access new data without redoing the whole retraining process or doing fine-tuning steps. In our case, this will enable the LLM to access the company data (company documentation) and use both

the questions from the user and the data as context to answer the specific queries regarding the company documentation (What Is LangChain?, n.d.).

**Vector Database**

While a vector database is not too dissimilar to a traditional database, it is much more efficient and more optimized to store vector embeddings of text data. Vector embeddings are data with large amount of dimensions for each word and help capture the semantic meaning of the words. Traditional databases find it quite challenging to deal with this large dimensional data, and hence a vector database is required to optimize not only the storage of the vector embeddings, but also the retrieval of the vectors for similarity search during the RAG process.

In traditional databases, the values of the observations that are queried exactly match the query from the user. In a vector database, the process is different since the data stored are in the form of vectors of numbers and not in the forms of strings or numbers.

Below diagram outlines the process of storing and retrieving vectors from a vector database.

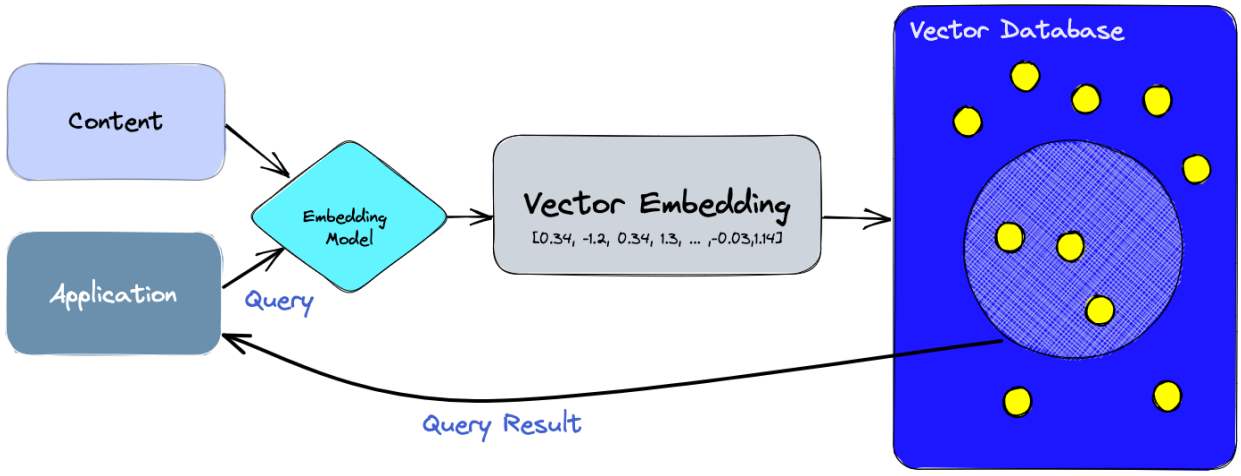



**Figure 1: Vector Database process (Pinecone)**

The first step consists of creating the embedding vectors of the data and inserting them into the vector database. When the user then issues a query, the same embedding model used to create the embedding vectors for the data is used to create embeddings for the query. The query embeddings are then used to search for similar embeddings from the vector database using an approximate nearest neighbor search and those nearest neighbors are finally retrieved from the database (Schwaber-Cohen, 2023).

**Retrieval-Augmented Generation**

LLMs are trained on huge amounts of data for various NLP tasks and are quite good at answering data in a general context. However, they also suffer from some weaknesses. For example, LLMs will invent a false answer even when it doesn't know the answer to the query or LLMs might provide outdated data since they are trained with data up to a certain data and therefore have no access to new data. This might be a problem in fields like medical or technology where new breakthroughs happen regularly.

Retrieval-Augmented Generation (RAG) is a process which addresses some of the weaknesses of LLMs and help improve the quality of the responses, especially in a specific domain or field. By implementing the RAG process, the LLM is provided with another database as reference and hence the LLM can generate more accurate and relevant responses in that domain as it uses both its training data and the new relevant data it has been provided as context to answer the queries. This process happens without needing to retrain the model and is therefore very cost-efficient, both in terms of time and computational and financial costs.

A summary of the RAG process is outlined below:

1. Without the process of RAG, the LLM simply generates its response based on the data that it was trained on. Therefore, the first step in RAG implementation is to create the external data source that will be provided to the LLM so that it can also infer from this data to generate the response. This external data can be made available in different formats and can come from different sources.
2. The second step is to create vector embeddings for the external data that is to be provided to the LLM. Since the LLM cannot process text in its initial text form and can only understand numbers, the external data has to be converted into numerical representations (vectors of numbers). These vectors are then stored in a vector database, which is an optimized version of a database for storing vectors.
3. The next step is to do a relevancy search. The query from the user is also converted to a numerical representation and this is matched with the embeddings of the external data stored in the vector database. The LLM will then retrieve the appropriate documentation from the vector database and compute a relevancy score for the query and the relevant retrieved data.
4. The LLM will then combine the query from the user and the relevant retrieved data from the vector database and this will serve as context to help the LLM generate more accurate responses (What Is RAG?, n.d.)

**References**

Word embeddings (2023): https://www.tensorflow.org/text/guide/word_embeddings.

What Is LangChain?, n.d. : https://aws.amazon.com/what-is/langchain.

What Is RAG?, n.d. : https://aws.amazon.com/what-is/retrieval-augmented-generation.

Schwaber-Cohen, R. (2023, May 3). *What is a vector database & how does it work? use cases + examples*. Pinecone. https://www.pinecone.io/learn/vector-database.

Touvron, Hugo, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, et al. “Llama 2: Open Foundation and Fine-Tuned Chat Models.” arXiv, July 19, 2023. https://doi.org/10.48550/arXiv.2307.09288.