POLITECNICO DI TORINO

Master's Degree in Data Science and Engineering

Master's Degree Thesis

Developing an Enterprise Chatbot using Machine Learning Models: A RAG and NLP based approach

Supervisors

Prof. PAOLO GARZA

Candidate

Fabio RIZZI

October 2024

Summary

The use of chatbots in companies is an effective solution for responding instantly to questions or problems, improving business productivity. Chatbots can be particularly useful within a company, where they can provide personalised assistance and speed up the process of retrieving information from documents and databases. However, accessing this data can be complex due to the fragmentation of information between disorganised business documents and structured databases that require technical skills to query, such as the use of SQL. This technical obstacle slows down access to information and makes work processes less efficient.Furthermore, it is not enough to be able to retrieve data: it is equally important to be able to visualise it in a clear and comprehensible way. Graphical visualisation, for example, makes it easier to grasp trends, anomalies and correlations, accelerating data understanding and decision-making. This project focuses on the creation of a chatbot to facilitate access to business information in Betacom s.r.l. , overcoming limitations related to the size of the company and the technical knowledge of the staff. The chatbot was developed to answer questions related to documents and databases in a simple and intuitive way, helping, for example, managers who are not familiar with SQL or other people who are looking for specific information without having to navigate between different departments.

The solution consists of a simple but functional interface, which allows queries to be made on both documents and databases. The main page allows the user to ask questions or access two sections: one for database searches with graphic visualisation and the other for tabular visualisation. The graphic search can return graphs (bar, line or pie charts) for immediate visual comparison. Interaction with documents, on the other hand, has been implemented as a chat, making the results more readable. Since this is a prototype, the two types of requests (documents and database) are separate, but their unification is planned in later stages. The back-end was developed in parallel with the front-end, which uses JavaScript and the Angular framework, with the application hosted on a private server. For the document part, the LLama 3 8B quantised 4-bit model was used for response generation, together with the FAISS database for semantic search with the addition

of a BM25 retriever for syntactic research. The pre-processing phase included transforming the documents into PDFs and dividing the texts into blocks to improve information retrieval. For SQL queries, the PipableAI model from hugging face was used to generate queries, with the help of an external framework, Vanna.ai, which helps in the search for tables to extract data from. The answers can be returned in the form of tables or graphs, depending on the user's needs.

In the evaluation of the retrieval phase, several metrics were analysed, including ROUGE for syntactic similarity and the BERT-score for semantic evaluation. However, a modified version of the Mean Reciprocal Rank (MRR), a metric that evaluates the position of the correct document in the result, was used. For the generation phase, a qualitative evaluation based on 3 possible grades was used: positive, negative or intermediate. To test the RAG system, a dataset consisting of internal documents provided by the company was used, while in the text-to-SQL phase, various models were explored. On the hardware side, the project was developed locally to reduce the costs associated with the cloud, this lead to choose open source language models such as Llama3 and Phi-3, for text generation, to reduce costs and ensure flexibility. Various techniques were tested to improve document searching and preprocessing. For example, techniques such as Hyde and Multiquery to generate variations of the query in order to expand the search range, or dividing the text by conditions on characters or text semantics.

In conclusion, this thesis demonstrated the importance and effectiveness of using advanced chatbots for information retrieval and management within complex business contexts. A key aspect of the project was the integration of advanced Retrieval-Augmented Generation (RAG) techniques, an approach not used in basic chatbot solutions. RAG techniques significantly improved the system's ability to retrieve accurate information from complex documents and corporate databases, enabling it to generate precise responses to even complex queries. Although the results obtained are promising, it is important to emphasise that the project is still a prototype. There are several margins for improvement, such as the smooth integration of document search and database query functionalities into a single interface and the further optimisation of responses to handle more complex requests. Furthermore, technological evolution will pave the way for new, more advanced artificial intelligence models and more user-friendly graphical interfaces. Ultimately, the enterprise chatbot developed in this project represents a significant step forward in the management and automation of business processes. The use of RAG techniques and the integration of LLM enable efficient and customised data retrieval, improving the quality and speed of daily operations. The potential of this technology is vast, and could be applied in many other contexts, helping to make organisations more agile, innovative and competitive.

Table of Contents

Chapter 1

Introduction

Imagine getting answers and solutions instantly for any doubt, without the hassle of searching or waiting. With their speed and availability, digital assistants (chatbots) use the user's laziness as their intent to comply. Software of this kind is not only useful for finding quick and effortless solutions; it is also an effective way of enhancing work productivity and optimising operations. In particular, having a specific chatbot within the company can provide personalised assistance and enhance the work experience of employees.

In today's business environment, the ability to retrieve information from documents and/or databases is essential to efficiently perform a wide range of daily tasks, regardless of the type of work. However, this process can be complex and cumbersome. This complexity can slow down the completion of daily tasks, increasing time. Consequently, it is very useful for companies to adopt solutions that simplify access to data, allowing information to be retrieved quickly and intuitively.

In order to automatize this process, a first issue is that finding the data needed to perform daily tasks in a company can be a complex process, as information is often scattered in different places, such as business documents or structured databases. Documents may be disorganised or stored in formats that make difficult the access and the search. On the other hand, business databases, although more structured, require specific skills to be queried effectively. For example, the use of SQL (Structured Query Language) is often necessary to extract data from these databases, but not all users have advanced technical skills, and navigating through this maze of data can be a daunting challenge. This technical barrier can significantly slow down information retrieval and make work more inefficient.

In addition, it is not enough to know where to look or how to extract data, it is equally important to be able to visualise it in an understandable format. In this context, the visual representation of information through graphs and dashboards proves crucial. Data visualised graphically can make it more immediate to understand trends, anomalies and correlations, which may not be evident through a

simple reading of tables or reports. This approach not only facilitates the interpretation of information, but also accelerates decision-making and task completion, improving productivity and quality of work. Companies that adopt advanced data retrieval and visualisation tools can therefore provide more intuitive and faster access to information, contributing to more effective and informed work in all its forms.

Considering the issues seen, a well-designed chatbot can revolutionise the data retrieval process within a company, simplifying access to both the information contained in documents and that stored in databases. Thanks to its intuitive interface, a chatbot can understand user queries in natural language and translate them into precise searches, without the need to know SQL or other advanced query techniques. It can extract relevant data from different sources, combine them if necessary, and present them in an immediately useful format: a simple text message for quick information or a graph to visualise trends and correlations more clearly. In this way, the chatbot becomes a powerful and accessible tool for all employees, facilitating work and improving business efficiency.
In particular, integrating machine learning into the chatbot could not only increase its efficiency and adaptability, but also enable it to better understand which action is the most appropriate to take based on the user's request. By analysing past interactions and identifying patterns in the data, the chatbot could anticipate user needs, simplifying the information retrieval process, and presenting the data in the most useful format, be it a text message or a graph.
In this context, large language models (LLMs) play a crucial role. LLMs, with their ability to understand and generate natural language text, enable the chatbot to interact more naturally and intuitively with users. These models, powered by huge amounts of data and advanced algorithms, can interpret even complex requests and translate them into concrete actions. Furthermore, LLMs, hardware limitations permitting, can quickly process large volumes of data, providing accurate and customised answers, further simplifying the entire process for the user.

This work is divided into the following chapters:

- Chapter 2: Related Work describes the state of the art and the fundamental concepts needed to understand and develop the chatbot, including embeddings, Retrieval-Augmented Generation (RAG) and Large Language Models (LLM).
- Chapter 3: Problem Statement and Proposed Solution presents the problem definition and proposed solution, illustrating the specific challenges faced by the company that the chatbot aims to solve.
- Chapter 4: Experimental Validation focuses on experimental validation, showing the tests performed to determine the optimal parameters, models and configurations in order to obtain the best results.

- Chapter 5: Conclusion contains the conclusions, summarising the results obtained and suggesting possible future developments.

Chapter 2

Related Work

2.1 Chatbot

A chatbot is a program that simulates a human conversation with a user. Over the years they have evolved, and we can now identify them into three main groups: Rule-based, AI-driven and Hybrid.

- Rule-based chatbots: These are the first types of chatbots to be created. When a user interacts with such a chatbot, the answers, that the program gives, are predefined and they are decided through a flowchart or a decision tree based on keywords or triggers.
  These types of chatbots are quick to develop and simple as they do not depend on machine learning algorithms but on 'if-else' statements. The main drawbacks are that the answers are predictable and limited only to specific scenarios.

- AI-driven chatbots: With the advent of new techniques in the field of machine learning, these types of chatbots have been developed. They are based on artificial intelligence and in particular on NLP -Natural Language Processing- algorithms that among them there is the generation of human-like text. Unlike the previous type, these are not rule-based and therefore the interaction are improved as they are more interactive. Machine learning algorithms are used to understand user input so that the most accurate text is generated.
  This kind of chatbot are more difficult to implement but offers more long-term benefits such as less maintenance.

- Hybrid Chatbots: Combining rule-based chatbot handling scenarios and machine learning algorithms from ai-driven chatbots has advantages regarding flexibility and adaptability but also disadvantages regarding complexity and

maintainability and the need to have competencies in both technologies. An example of a famous hybrid chatbot is RASA.

Each type of chatbot has its advantages and disadvantages, but AI-driven chatbots stand out for their ability to offer more dynamic and responsive conversations. Thanks to machine learning algorithms and NLP, these chatbots can adapt to a wide range of interactions, responding more naturally to any user query or message without being confined to predefined rules. Even though this approach requires trusting in the capabilities of the machine learning model, choosing AI-driven chatbots would be the best option for those seeking a more fluid and spontaneous interaction.

2.2 Embeddings

An embedding is a numerical representation of a piece of information, such as texts, documents, images, audios, and so on. The representation captures the meaning of what is being embedded, making it manageable for many applications. Thus, an embedding of a sentence or other type can be represented in a vector space so that the embedding can be confronted with each other, in particular when the embedding represents the semantic meaning of a text, the closer the embeddings, the more similar the texts. This is at the root of the most of NLP algorithms, from recommendation systems to chatbots. The vectors in an embedding representation can be divided in two groups : Dense and Sparse Vectors.

2.2.1 Sparse Vector

Type of representation where most of the elements are zero. They often have a very high dimension because it depends on the size of the vocabulary of the text. It is the simpler method and also the memory usage is less than other method because only the non-zero elements are stored. There are different methods to build a sparse vector from a text. The most common ones are:

Bag-of-Words (BoW)

It is a model used to represent a text as a disordered set of words of which the number of occurrences is saved. The numbers of occurrences create the vector.

Term frequency–inverse document frequency (TFIDF)

It is an improvement on the simple bag-of-words model in which weights are defined for each word based on the entire text. It is the product of two statistics: term

frequency and inverse document frequency, the product defines the importance of a word within the text with respect to a set of texts.

TF(t,d) represents how often a word appears in a single document

$$TF(t,d) = \frac{\text{Number of occurrences of term} t \text{ in document } d}{\text{Total number of terms in document } d} = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}}$$

IDF(t,D) reduces the weight of common words in many documents, increasing the importance of more specific words.

$$\text{IDF}(t) = \log\left(\frac{|D|}{|\{d \in D : t \in d\}|}\right)$$

Thus, the product is:

$$\text{TF-IDF}(t,d,D) = \text{TF}(t,d) \times \text{IDF}(t,D)$$

Where

- t=term
- d=document
- D=set of documents

BM25

This is a ranking algorithm because it estimates the relevance of documents based on a query, but in the process, it uses a vectorization.
Given a query Q containing keywords $q_1, q_2, \ldots, q_n$, the BM25 score of a document D is:

$$\text{score}(D,Q) = \sum_{i=1}^{n} \text{IDF}(q_i) \cdot \frac{f(q_i,D) \cdot (k_1+1)}{f(q_i,D) + k_1 \cdot \left(1 - b + b \cdot \frac{|D|}{\text{avgdl}}\right)}$$

$$\text{IDF}(q_i) = \ln\left(\frac{N - n(q_i) + 0.5}{n(q_i) + 0.5} + 1\right)$$

where:

- D = Document
- Q = Query (set of terms $q_1, q_2, \ldots, q_n$)
- $\text{IDF}(q_i)$ = IDF weight of the term q_i
- $f(q_i, D)$ = Number of occurrences of term q_i in document D

- $|D|$ = Length of document D in words
- avgdl = Average document length in the collection
- k_1 and b = Free parameters (typically $k_1 \in [1.2, 2.0]$ and $b = 0.75$)
- N = Total number of documents in the collection
- $n(q_i)$ = Number of documents containing the term q_i

Explanation of most important part:

- Term Frequency $f(q_i, D)$: Counts how many times a term q_i appears in a document D. The more frequently a term appears in a document, the more relevant it is for that document.
- Document Length $|D|$: Length of the document in terms of the number of words. Used to normalize the term score.
- Average Document Length avgdl: The average length of documents in the collection. Used to adjust the score to account for the average document length.
- IDF: Measures the importance of a term based on its frequency in the documents. Terms that appear rarely have a higher IDF, indicating they are more significant.
- Parameters k_1 and b:
  - k_1: Adjusts the effect of term frequency. Higher values increase the impact of term frequency.
  - b: Adjusts the effect of document length. Higher values make the score more sensitive to document length.

Sparse vector method were the first to be developed and can transform a text into a vector based on words, but not on the semantic value of the text.

2.2.2 Dense Vector

Unlike sparse representation, dense vectors can have multiple dimensions and collect the semantic relationships between words and richer contexts. We can divide these methods into 2 different groups: Static and contextual models.

Static embedding

Static embedding models represent words by means of vectors that do not change according to context. These models use different approaches to generate such vectors but the representations remain fixed and do not take into account variations in the meaning of words depending on the sentence in which they appear.
Some important examples of methods are:

- Word2Vec, developed by Google, is one of the pioneering models for creating dense vectors. It uses approaches such as Continuous Bag of Words (CBOW) and Skip-Gram to transform words into vectors. Semantically similar words have neighboring vectors in space, but the representation of a word is invariable, regardless of its usage. Variants of word2vec have been generated.

- FastText: A variant of Word2Vec developed by Facebook, FastText represents words as compositions of n-grams of characters. This approach improves the handling of rare and derived words compared to Word2Vec.

- GloVe (Global Vectors for Word Representation): This model, developed by Stanford, relies on the analysis of co-occurrences of words in a text corpus to generate vectors that reflect global semantic relationships.

Contextual models

Contextual models represent a significant advancement over static models. These models dynamically update word representations according to the context of the sentence, enabling a finer and more precise understanding of meaning. The most prominent example is certainly BERT (Bidirectional Encoder Representations from Transformers): developed by Google, it represents an innovative Transformer-based approach. It provides bidirectional contextual vectors of words, considering them in both the previous and the next context. This allows BERT to capture more precise meanings adapted to the specific context of the sentence, significantly improving language understanding compared to static models. Variants were also generated from it.

- RoBERTa: Optimises BERT's training to improve performance. It uses training on a larger corpus and advanced optimisation techniques for superior results.

- DistilBERT: A lighter version of BERT, designed to retain much of BERT's representation capabilities but with reduced size and computational complexity, making it more efficient.

2.2.3 Usage

New models, especially contextual models as the others are becoming obsolete, are coming out all the time. Examples are those classified in the Hugging Face MTEB (Massive Text Embedding Benchmark) Leadboard, which continue to push the boundaries of text representation. These models offer increasingly sophisticated solutions for a wide range of NLP applications.
For the use of these methods, which continue to evolve rapidly, there are industry-leading libraries that make the integration of these technologies as simple as possible. Of these, Hugging Face's Transformers library is particularly renowned, offering a wide range of Transformer-based models. In addition, Sentence Transformers, another Hugging Face library, is specifically designed to facilitate the creation of vector representations of sentences, making it easy to apply advanced text analysis techniques in real-world applications. These libraries allow developers to exploit the most advanced models in the field of NLP without having to deal with the complexity of implementation from scratch.

2.2.4 Application

In the field of text representation, it is essential to consider both the semantic dimension, which focuses on the meaning of words, and the syntactic dimension, which focuses on structure and keywords. For a complete and accurate understanding of language, the integration of both sparse and dense representations enables the capture of deeper nuances, thus improving the quality and effectiveness of NLP applications.

2.2.5 Similarity Metrics

In research on dense and sparse vectors, evaluating vector similarity is crucial for NLP tasks such as information retrieval and document similarity. Three key metrics used for this purpose are cosine similarity, Euclidean distance, and maximum inner product.

- **Cosine similarity** measures the angle between two vectors and is calculated using the formula:

$$\text{Cosine Similarity} = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|}$$

  where $\mathbf{A}$ and $\mathbf{B}$ are the vectors being compared, $\mathbf{A} \cdot \mathbf{B}$ is the dot product, and $\|\mathbf{A}\|$ and $\|\mathbf{B}\|$ are their norms. This metric is useful for assessing how aligned two vectors are in the same direction, regardless of their magnitude, and is often used to measure semantic similarity between words or documents.

- **Euclidean distance** measures the linear distance between two vectors in multidimensional space and is calculated using the formula:

$$\text{Euclidean Distance} = \sqrt{\sum_{i=1}^{n}(a_i - b_i)^2}$$

where a_i and b_i are the component values of vectors $\mathbf{A}$ and $\mathbf{B}$, and n is the number of dimensions in the vector. This metric provides an absolute measure of the distance between the two vectors, useful for identifying overall differences in their representations.

- **maximum inner product** is calculated as:

$$\text{Inner Product} = \mathbf{A} \cdot \mathbf{B} = \sum_{i=1}^{n} a_i \cdot b_i$$

where a_i and b_i are the components of vectors $\mathbf{A}$ and $\mathbf{B}$. This metric directly considers the dot product, measuring similarity in terms of alignment and intensity, and is particularly useful in contexts like recommendation systems, where both the direction and magnitude of the vectors are important.

2.3 LLM

Large Language Models (LLMs) represent one of the most significant innovations in the field of artificial intelligence and natural language processing (NLP), with the ability to understand, generate and even interact in human language with a good level of satisfaction. These models are the result of years of research and development, culminating in the creation of advanced architectures that exploit billions of parameters and vast text datasets to learn the complexity and richness of natural language. In training LLMs, models are exposed to huge amounts of unlabelled text and learn to predict the next token in a sequence, thus gaining a deep understanding of linguistic structures and semantic relationships.

A key element of the power of these models is their adaptability: through techniques such as fine-tuning, in which the model is further trained on a specific dataset to optimise its performance on a particular task, or prompting, which guides the model to produce more relevant and contextually relevant responses, LLMs can be customised to address a wide range of application needs.

The architectures on which LLMs are based are varied and sophisticated. Autoregressive models, such as those in the Generative Pre-trained Transformer (GPT) family, are designed to predict the next token in a sequence based on previous tokens, thus creating a chain of text that can range from simple sentence completion to the

generation of entire paragraphs of content. On the other hand, encoder-decoder models, such as BERT (Bidirectional Encoder Representations from Transformers) and T5 (Text-To-Text Transfer Transformer), use an encoding-decoding structure that allows them to handle complex tasks such as answering questions, summarising text and translating, modelling language bidirectionally and understanding the full context of a sentence or document.

In addiction, it is crucial to consider the context window of the models. The context window refers to the maximum number of tokens (i.e. units of text such as words or portions of words) that the model can take into account simultaneously when processing an input. Monitoring and managing the context window correctly is therefore essential to ensure that the model receives all the information it needs to generate the correct output.

Despite their extraordinary capabilities, LLMs are not without limitations. One of the main challenges is the tendency to produce 'hallucinations', i.e. responses that, while seeming coherent and sensible, are in fact incorrect or invented. This phenomenon is particularly problematic in critical applications, where accuracy of information is crucial. Furthermore, LLMs can amplify biases in training data, reflecting and sometimes exacerbating social or cultural biases. The training and execution of these models also require huge amounts of computational resources, which raises questions regarding environmental impact, as the energy consumption associated with their execution is significant. The research and development of LLMs is constantly evolving. Every year, new models and techniques are introduced that further push the boundaries of what is possible with natural language. Platforms such as Hugging Face offer an up-to-date overview of the best open-source models available, allowing researchers and developers to explore and utilise the latest innovations in the field. This continuous innovation not only expands possible applications, but also seeks to address current limitations and challenges, making LLMs increasingly efficient, reliable and applicable in real-world contexts. Ultimately, Large Language Models represent a crucial frontier in artificial intelligence, with the potential to radically transform the way we interact with technology and the world around us.

2.4 RAG

To adapt a Large Language Model (LLM) to its specific task, there are several strategies such as fine-tuning, prompting and Retrieval-Augmented Generation (RAG). In this section, we will focus on the latter technique, which combines information retrieval with content generation to provide more precise and contextualised responses. RAG works through a multi-stage process. Initially, the user's query

is transformed into a vector via an embedding model, which semantically represents the query. This vector is then compared with a vector database containing semantic representations of various blocks of text called chunks. The most relevant contexts are retrieved, measured by the distance between the query vector and the embeddings in the database. Subsequently, both the original query and the retrieved contexts are passed to the LLM, which uses this information to generate a customised response. This response can be provided together with links to the sources of the information used, thus guaranteeing transparency and reliability. Thus, RAG represents a powerful method of harnessing the capabilities of LLMs, integrating external data with content generation to answer complex questions in a precise and informed manner.

The Retrieval-Augmented Generation (RAG) technique has several significant advantages, making it a powerful choice for improving the effectiveness of LLMs in various contexts. One of the main benefits is access to up-to-date and reliable information. Unlike models that rely solely on training data, which can quickly become outdated, RAG allows the integration of recent and relevant knowledge, ensuring more accurate and contextualised responses. Furthermore, RAG significantly reduces the risk of sensitive data leakage, as sensitive information does not have to be included in the model's training data, but can be securely and contextually retrieved at the time of the request. This approach also helps reduce the computational and financial costs associated with LLM-based applications. Since the model does not need to be continually re-trained to include new information, RAG lowers the need for computationally intensive resources, making the implementation of LLM in enterprise environments more sustainable. Finally, this technique also succeeds in decreasing an issue present in LLMs that is 'hallucinations', especially if it is implemented in such a way that it only takes information from vector databases and not from prior knowledge of the model. However, like any technology, RAG also presents challenges. The quality of answers depends on the quality of external sources and the system's ability to retrieve the most relevant information, which can be a limitation in environments with poorly structured or low quality data.

2.5 Prompting

Another fundamental technique for the effective use of large language models (LLM) is prompting. There are several prompting techniques that can be used to guide the model to respond in the desired manner. Of these, three of the most common are zero-shot prompting, few-shot prompting (and its one-shot variant), and chain of thought prompting.

Zero-shot prompting

Zero-shot prompting means that the prompt used to interact with the model contains no examples or demonstrations. The model receives a direct instruction to perform a task, with no additional examples provided to direct it. This type of prompting relies solely on the model's ability to understand and generalise from the given instruction. However, when zero-shot prompting does not produce satisfactory results, it may be useful to switch to more elaborate techniques.

Few-shot prompting

Few-shot prompting comes into play when it is necessary to provide examples or demonstrations to improve the model's performance. This technique allows contextual learning, where examples are inserted into the prompt to condition the model to generate more accurate and relevant responses. Demonstrations serve as a guide, helping the model to better understand the task at hand. For example, for a text completion task, we might include in the prompt some sentences that have already been completed to make the model understand how it should proceed. There is also a variant called one-shot prompting, where only one example is given. Although these techniques are very effective, they may still be insufficient for tasks requiring more complex reasoning.

Chain of thought prompting

To tackle such tasks, chain of thought (CoT) prompting can be used. This technique allows the model to develop complex reasoning skills through intermediate steps of thought. In practice, instead of simply providing a direct answer, the model is guided to perform a series of logical or argumentative steps before arriving at the final answer. Chain of thought prompting can be combined with few-shot prompting to achieve better results in tasks requiring more complex reasoning before providing an answer.

2.6 Text-to-SQL

The Text-to-SQL task is an area of research in Natural Language Processing (NLP) that aims to automatically generate SQL queries from natural language text. This task requires the conversion of textual input into a structured representation, which is then used to generate a semantically correct SQL query, ready to be executed on a database.

One of the main obstacles in the text-to-SQL task is the different dialects of SQL used in the various database management systems (DBMSs). Each DBMS, such as MySQL, PostgreSQL, SQL Server, and others, implements SQL with

slight variations and peculiarities, making it complex to generate queries that are compatible with different systems. The ability to adapt to these differences is crucial for generalising the model and ensuring the accuracy of queries generated on different platforms.

Another significant challenge is the complexity of the queries that the system must be able to generate. Simple queries, involving direct data selections and basic filters, can be handled with relative ease. However, more complex queries, which require joins between multiple tables, sub-queries, advanced aggregations and custom functions, pose a greater problem. Such queries require not only a thorough understanding of the database structure, but also a sophisticated ability to correctly represent the logic required by natural language in an executable SQL form.

These factors make the text-to-SQL task a fascinating and complex challenge, with a significant impact in the area of human-computer interaction and in improving data accessibility through natural language.

In practice, besides the academic approaches and commercial products of large companies, including the popular CHATGPT, there are also online platforms that offer Text-to-SQL services, allowing users to automatically generate SQL queries from natural language input. For instance, AI2SQL and Draxlr are platforms that offer an intuitive interface to automatically create and visualise SQL queries, simplifying access to data for analysts and business intelligence professionals

Chapter 3

Problem Statement and Proposed solution

3.1 Problem Statement

Betacom is an innovative IT company specialising in consulting, design, prototyping and development of IT solutions. During my time in the company, I had the opportunity to work on a project that met a specific need: to simplify and improve the process of access to internal company information for non-technical employees. The main problem encountered within Betacom and its partner companies is related to the difficulty of obtaining technical or management information, especially for employees without IT skills. For example, many operations that require the use of query languages such as SQL or knowledge of technical procedures are difficult for people working in areas such as purchasing, human resources or administration. This leads to a loss of time and resources, as employees have to go to technical departments to obtain information or solve problems.
Furthermore, the organisational complexity of the company, with the presence of partner and acquired companies, makes it difficult for many employees to know where to turn for specific information, further increasing the time spent on research and communication activities.

The aim of the project is to develop a chatbot that facilitates access to company information through a simple and intuitive interface. This tool must:

- Allow non-technical employees to easily obtain technical or business information without having to know programming languages or complex procedures.
- Reduce the time spent searching for information by automating processes that would normally require the intervention of an IT expert.

- Facilitate communication between partner and acquired companies, ensuring that information can be obtained quickly and accurately, regardless of department or sector.

A concrete example could be a purchasing department manager who needs information on a transaction stored in the database, but does not have the skills to execute SQL queries. The chatbot will be able to understand the query in natural language and return the necessary information, greatly simplifying the operation.

3.2 Proposed solution

For the aesthetics of the interface, which is not my strong point, much less the main point of this thesis, , we opted for a basic but nevertheless pleasant version, focusing more on the practicality of the interface. The chatbot comprises a main page on which one can already write questions about documents, or one can go via links to the other two important pages. The first page allows the query to the database via a graph, following the first link. The second page, on the other hand, allows data to be displayed on a tabular basis, following the second link. Queries based on information from the database are presented in two different options depending on the type of search to be performed. The first option concerns a static search for information in which only the desired list is to be obtained; in this case, the output is by means of a table, also because the visualisation in written format was not as visually appealing. In the second option, the output is graphical, using bar, line or pie charts, so that the returned data can be compared immediately. This is particularly useful when you want to get a general idea rather than focusing on specific data. The document query part was constructed as a chat, to return data in a more readable and clearer format than a simple text extracted from a file.

Since this is a prototype of the project, it was decided to separate the two types of requests, the one on documents and the one on the database, since the unification, which will take place at a later stage, will in any case be feasible and relatively simple compared to the creation of these two separate parts. Requests based on information from the database are designed to be displayed in two different ways, depending on the user's needs. Therefore, in parallel with the development and testing of the backend in order to achieve maximum results, the minimal front-end code necessary to create the graphical user interface was also developed. The code was written in javascript using the angular framework, hosted in a private server.

Here are the images of the main parts.

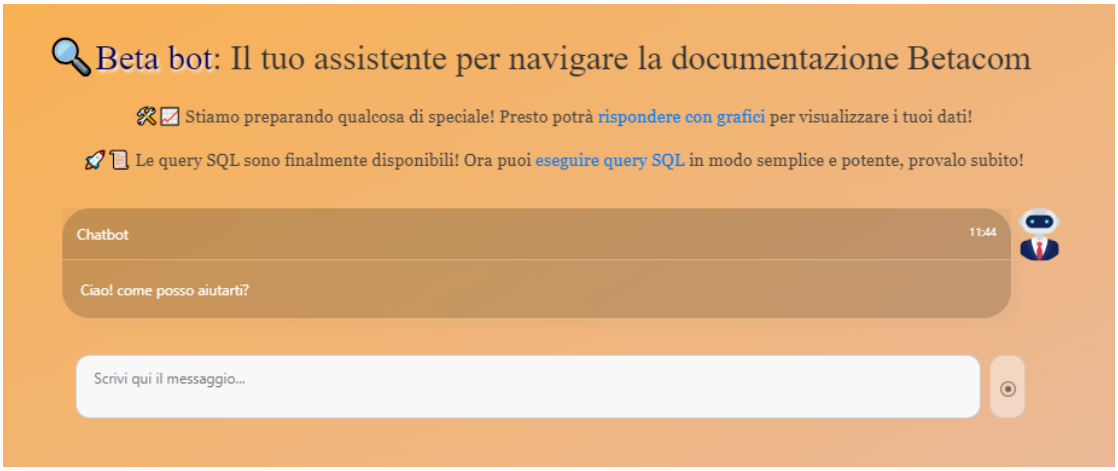

Figure 3.1: main page of the chatbot

Figure 3.1 shows the main interface of the chatbot at start-up. This image illustrates the start-up page of the system. From here one can start using the three main components: the immediately accessible user interaction section for searching for data in documents, the part dedicated to returning data in table format accessible via the second link, and the part for generating graphs accessible via the first link.

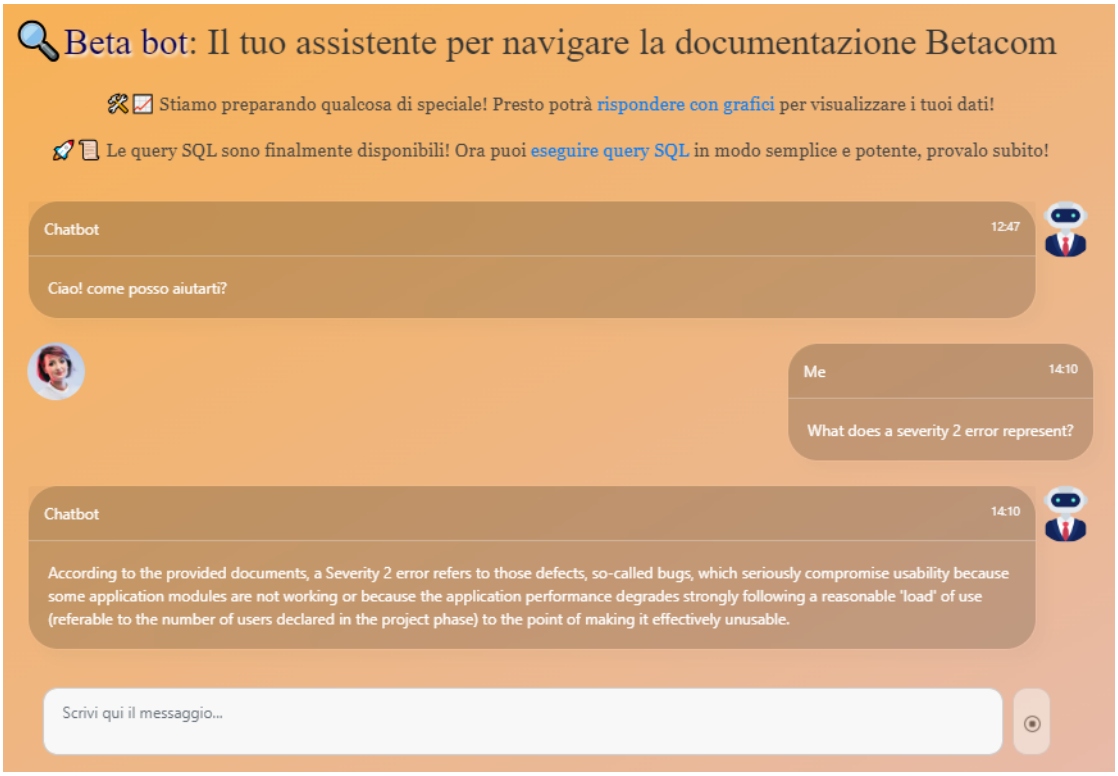

Figure 3.2: Main page of the chatbot after a request is sent

Figure 3.2 highlights the section of the chatbot used to extract information from documents after a query has been submitted. This part is designed to allow the user to query various previously processed and saved documents.

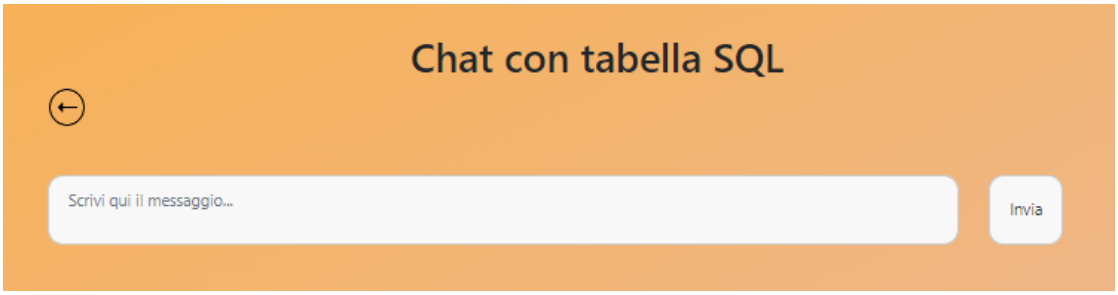

Figure 3.3: Page about tabular query result

Figure 3.3 shows the section of the chatbot that returns data in tabular format, which is accessed using the link. In this figure it is shown at the stage where it is

ready to listen to the request.

Figure 3.4: Page about tabular query result after a request is sent

Figure 3.4 shows the page of the chatbot returning the tabular data at the data return stage. In the event of an error, a message is displayed instead of the table. Below the table is the text input again, as the page is always ready to accept requests. This table returned the data concerning the query: 'For each song of genre 'Alternative' show me the name, genre, duration and album name'.

Figure 3.5: Page about chart query result

Figure 3.5 shows the section of the chatbot that returns the data in graphical format, which is accessed using the other link on the home page. In the request waiting phase, which is shown, there are input buttons to send the request and select which type of graph you want it in.

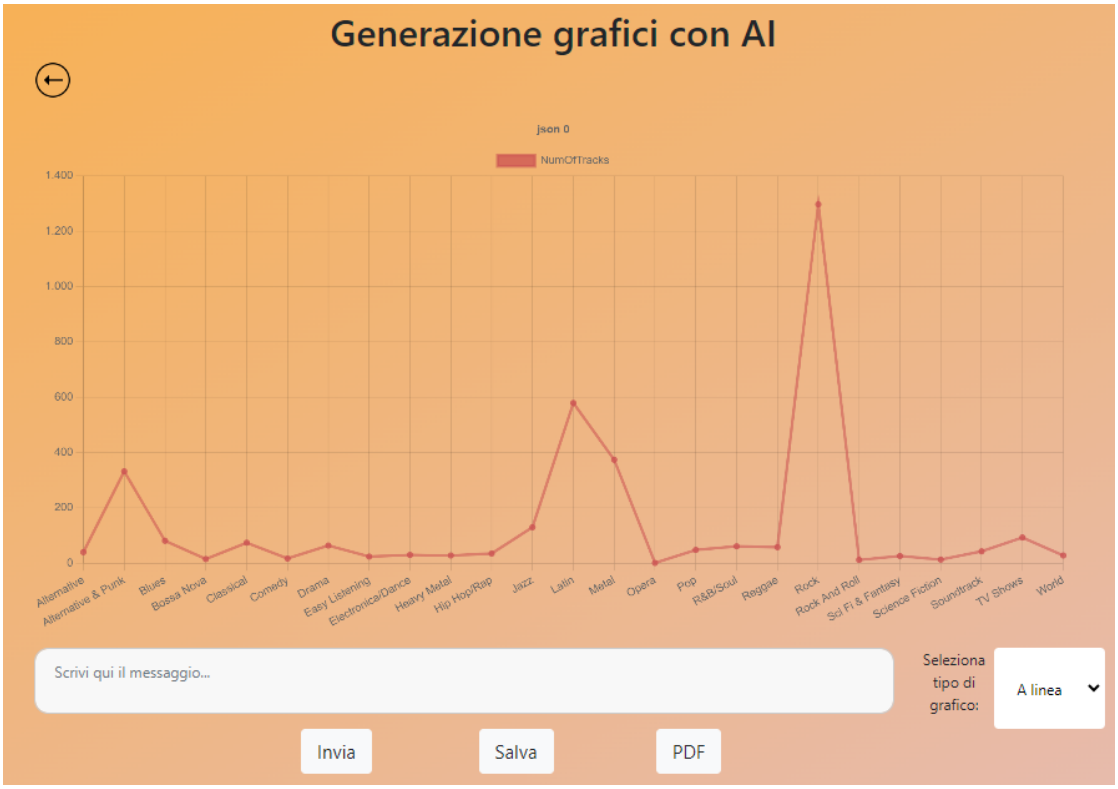

Figure 3.6: Page about chart query result after a request is sent and 'line' is selected

Finally, Figure 3.6 and 3.7 show the chart format section after receiving the response. At this stage it is possible to change the chart type in the same way as before and use the input buttons to download the chart. In this graph, data is shown from the request: 'For each genre show me the number of songs'.

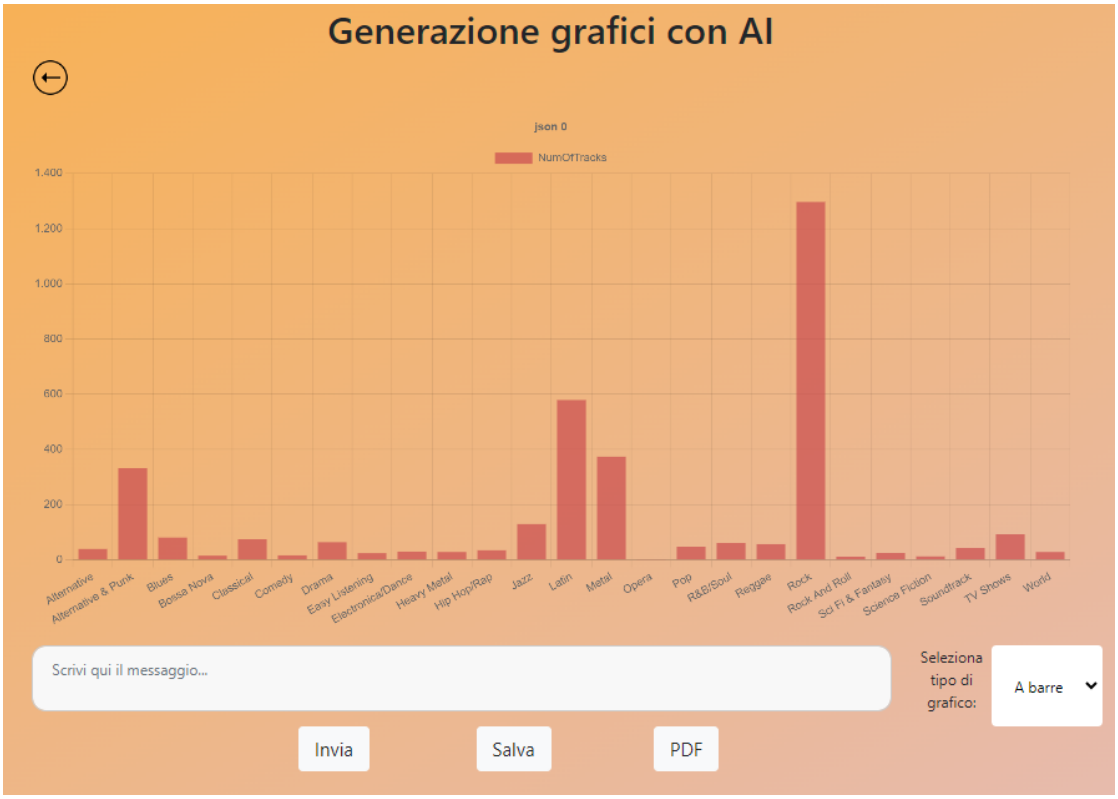

Figure 3.7: Page about chart query result after a request is sent and 'bar' is selected

Getting more technical, various tests were carried out to arrive at the combination of models, parameters and techniques used in the final solution. The tests and the choices made will be addressed more extensively in the next chapter. At the end of the test, the following final decisions were reached.
The final version for the **document part** consists of the use of LLama 3 8B quantized to 4 bits as the main LLM that deals with the generation of the response from data taken from the database. A model available on github Alibaba-NLP/gte-large-en-v1.5 was chosen as the embedding model. For document retrieval, FAISS was used as a vector database and both FAISS semantic search with a metric based on Euclidean distance and keyword search implemented using the BM25 metric were used. MMR was used as the search technique and at the end of it all, a reranking based on a model also taken from huggingface called bge-large was implemented. As pre-processing, these steps were taken:

- Transformation in PDFs
- Semantic division of the chunks using a per-character division if the 5000 characters are exceeded

- Replacement of abbreviations
- Expansion of dates
- Addition of the document name for each chunk
- English translation of each block of text to embed

No particular technique was used for generation, such as compression, but only a specific prompt, this one:

“’You are a helpful assistant that extracts informations from a given text. I will give you 200 $ if you respond in a detailed manner and verify that the answer contains all the information requested by the question without adding anything unnecessary.
Follow these steps:
Carefully read the question.
Carefully read the answer.
CHECK THAT THE ANSWER CONTAINS ALL THE INFORMATION REQUESTED BY THE QUESTION AND NOTHING MORE.
Ensure that the answer does not contain any additional, unnecessary information and that the information in the answer is not contradictory.
Only answer to the given question with the information you can find here:"’

For the **SQL part**, a model taken from huggingface called PipableAI/pip-library-etl-1.3b is used. To help with the generation of SQL queries, RAG is also used here. It was not implemented from scratch but using the Vanna.ai framework. The preprocessing for this is based only on retrieving the ddl of the datatbase.

For the model there is a specific prompt with which the model was trained.
This is the prompt used for returning the table result:
"""Generate a simple SQL query from the schema mentioned for the following question.
If you don’t find the a possible query, return SELECT ’cinoooo’ AS response;
<schema>schema</schema>
<question>question</question>
<sql>"""

This the pronpt used for returning the chart :
"""Generate a simple SQL query from the schema mentioned for the following question.
If you don’t find the a possible query, return SELECT ’cinoooo’ AS response;

If there are only 2 column, then you must put the more important one as second one.
<schema>schema</schema>
<question>question</question>
<sql>"""

This piepeline is used for both the tabular and the graphical part as the same resources are used and differ only in the type of output.

Chapter 4

Experimental validation

4.1 Metrics

In this section, I will examine in detail the metrics used for the experiments described in the following paragraphs. Considering that Retrieval-Augmented Generation (RAG) and the text-to-SQL task are difficult to evaluate objectively, most of the metrics adopted tend to be qualitative rather than quantitative in nature.

4.1.1 Metrics for retriever test

With regard to Retrieval-Augmented Generation (RAG) experiments, particularly in the retrieval phase, the result of these tests is to return a number n of document fragments, called chunks, most similar to the requested query. The evaluation then focuses on the similarity between the returned chunks and the expected chunk to return. Although one can measure this similarity with objective values, such as the Rouge score, these methods however only evaluate syntactic similarity by comparing similar words without considering the meaning of the documents. One could also rely to metrics based on machine learning models, such as the BERT-score. However, this would imply evaluating one model using another, which, despite the irony, we decided not to do. Therefore, we started with a metric called Mean Reciprocal Rank (MRR), which evaluates the position in which the expected chunk was returned, and modified it slightly, making it less objective but more suitable for measuring the result qualitatively. Below, I present the metrics mentioned and the version chosen.

Rouge

ROUGE (Recall-Oriented Understudy for Gisting Evaluation) is a metric commonly used to assess the quality of summaries and other automatically generated texts by comparing them with reference texts. It is based on counting the coincidences of n-grams (word sequences), sentences, and units of variable length between the generated text and the reference text. ROUGE-1, ROUGE-2 and ROUGE-L are the most common variants, measuring the overlap of single words, consecutive word pairs and the longest common sub-sequence, respectively. However, this metric focuses on syntactic similarity, thus assessing how much the texts share similar words, without taking into account the overall meaning.

Bert

BERT-score is a more advanced metric than Rouge because it uses language models such as BERT (Bidirectional Encoder Representations from Transformers) to assess similarity between texts at a semantic level rather than just a syntactic level. Rather than simply comparing exact words, BERT-score considers the word vector representations (embeddings) generated by BERT, thus capturing the similarity of meaning between texts. This metric is particularly useful for evaluating texts in which synonyms or paraphrases could be used, better reflecting the semantic understanding of the content.

MRR

Mean Reciprocal Rank (MRR) is a metric used primarily in the context of information retrieval and recommendation systems to assess the quality of an ordered list of results. MRR is calculated as the average of the reciprocals of the ranks of the first correct answers returned by a system in a series of queries. In other words, it considers in which position the first relevant result is found for each query and averages these values over all queries tested. MRR is useful for understanding how quickly a system can provide a useful answer, but in some contexts it may be considered too focused on specific positions, neglecting a more overall assessment of the quality of results.

CHOSEN METRIC

To measure the retrieving process, a metric similar to MRR was chosen. In this case, if a chunk is not completely correct but is close to the correct answer, such as a chunk that still contains the answer but not the default answer, or a chunk that only partially includes the answer, the question receives a slightly lower score than it would with the MRR for a query. Instead, if the chunk is totally correct, the

value of the query is the reciprocal of the rank.
Furthermore, to simplify the presentation of the results, I will show two values: one for the percentage of correct chunks returned as the first block and the other for the percentage of correct chunks returned within the first 4. Thus, these values will correspond to an MRR of 1 and an MRR other than 0, respectively.

4.1.2 Metrics for generation test

As far as the generation phase of RAG is concerned, objectively evaluating the responses of a large language model (LLM) is even more complex. There are methods such as RAGAS, which uses GPT to compare the responses generated by my pipeline with those produced by GPT, both for generation and retrieval. However, even in this case, one would rely to evaluating a machine learning pipeline using another machine learning model, with all the limitations and ironies of the case.
To overcome these difficulties, we chose to use questions with known answers as tests, evaluating the answers generated with three possible values: "Correct", "Uncorrect" and "quasi-correct" . 'Correct' indicates a correct and well-formulated answer, "Uncorrect" a completely incorrect answer, while "Quasi-correct" represents an intermediate evaluation, such as an answer that contains correct information but is inaccurately presented, or an answer that is well formulated but omits part of the relevant information.

4.1.3 Metrics for text-to-sql test

For the text-to-SQL task, several queries will be made on a test database. As metrics, both the percentage of queries exactly matching the correct ones and metrics based on the number of rows returned from the table compared with those returned by the correct query will be used.

4.2 Dataset

4.2.1 RAG Dataset

To test the RAG part, I needed a large number of documents and questions relevant to these topics. After initially using basic documents found online, the company provided me with access to internal documents, which I used to generate the dataset needed to test my code. The use of internal documents was crucial to the project, as it is precisely these types of documents that will be used in the final version. In particular, I worked with:

- Documents: 50 documents concerning customer agreements for the creation of software and project presentation documents.
- Questions: From these documents, I initially generated 20 questions for a first round of testing, but realising that these were not enough, I created further questions, reaching a total of 35.

4.2.2 Embedding Dataset

For the choice of the embedding model to be used, the RAG dataset on which all pipeline tests were conducted was not used. Two documents were used to test these types of models: one in Italian and one in English.

For these documents, 30 questions were formulated for use during the tests.

4.2.3 Summarization Dataset

To evaluate the performance of the large language models (LLM) used in the study, a dataset widely known in the field of automatic summarisation was used: BBC News, available on Kaggle. This corpus, specifically designed for extractive summarisation, comprises 417 BBC political news articles published between 2004 and 2005. For each article, five reference summaries are available.[1]
In order to focus the analysis on a specific domain, a subset of the dataset consisting exclusively of technology articles was selected, with a total of 30 documents. Of these, 20 were retained in the original language (English) to assess the models' ability to process and summarise English-language texts. The remaining 10 articles were translated into Italian, allowing us to test the multilingual abilities of the models and their ability to generate accurate summaries in both languages.

4.2.4 Dataset text-to-sql

For the text-to-SQL task, two test databases are used: one in PostgreSQL and the other in SQLite, commonly used as a test database, known as Chinook.

4.3 HW Architecture

The project was chosen to be developed locally, thus avoiding the costs associated with creating a cloud version, such as hosting subscriptions and the use of cloud resources on platforms such as AWS or Microsoft Azure. This decision required the hardware limitations of the machine on which this software was developed to be taken into account several times in the design choices.

Hardware specifications: The machine specifications include a Linux operating system with the Rocky Linux distribution, 64 GB of RAM, two AMD Radeon RX 6800 GPUs of 16 GB and a 1 TB hard disk drive.

4.4 RAG

4.4.1 CHoosing the LLM

In order to maintain the approach of a local version and reduce costs, it was decided to use only open source or open weight models. This choice was facilitated by the increasing availability of large language models (LLM) released under open source licences or with accessible weights, many of which are available on Hugging Face, a leading platform for sharing machine learning models. Due to the variety and accessibility of these models, several were tested from Hugging Face to identify the most suitable ones for the project's needs, while ensuring high quality without incurring additional costs for licences or subscriptions.

LLAMA3

Meta has released Llama 3, one of the latest iterations of its family of open-access language models, available on Hugging Face. This family comprises two main variants: the 8 billion (8B) model and the 70 billion (70B) parameter model. Both models are offered in pre-trained and instruction-optimised versions designed to enhance dialogue applications. Llama 3 uses an advanced auto-regressive transformer architecture, and the trained versions have been refined through supervised fine-tuning (SFT) and reinforcement with human feedback (RLHF) to better align with human preferences in terms of utility and safety. The model is trained on over 15 trillion tokens of publicly available data, with a context length of 8,000 tokens and a new tokenizer that expands the vocabulary to 128,256 tokens. The educated variants are particularly suitable for conversational scenarios and show superior performance compared to many current open source models on common industry benchmarks. The 8 billion parameter version, that we choose to test, is just over 16 GB in size. Llama 3 was released on 18 April 2024 and can be used via Hugging Face's Transformers library, with support for inference and fine-tuning.
To interact with Llama 3 models, the following prompt type is used:
"""<|begin_of_text|><|start_header_id|>system<|end_header_id|>.
System phrase to make the model understand its task and situation. <|eot_id|>
<|start_header_id|>user<|end_header_id|>
User text <|eot_id|><|start_header_id|>assistant<|end_header_id|>"""

PHI3

Phi-3 is a family of innovative language models, among which the Phi-3-Mini, a 3.8 billion parameter model, stands out. Despite its relatively small size, Phi-3-Mini demonstrates a high level of performance, comparable to larger models such as Mixtral 8x7B and GPT-3.5. This model was trained on a vast dataset of 3.3 trillion tokens, consisting of filtered web data and synthetic data, which enhances its robustness, security and suitability for the chat format. The main novelty of Phi-3 lies in its innovative approach to the training dataset, which is an evolution from that used for Phi-2. In addition, Phi-3-Mini is designed to be light enough to be implemented on mobile devices, with a size of just under 8 GB. It is available in two context length variants: 4K and 128K tokens, with the Phi-3-Mini-4K-Instruct demonstrating outstanding performance in areas such as reasoning, language understanding and logic. The Phi-3-Mini-4K-Instruct, in particular, has undergone post-training refinement that includes both supervised fine-tuning and direct preference optimisation, further improving its instruction-following capabilities and providing advanced safety measures. When evaluated on benchmarks testing common sense, language understanding, mathematics, code and logical reasoning, Phi-3-Mini proved to be among the best in its class of models with less than 13 billion parameters. For the technical implementation, Phi-3-Mini uses a tokenizer similar to that of Llama, with some additional token extensions. In addition, the model employs advanced techniques such as Phi3SuScaledRotaryEmbedding and Phi3YarnScaledRotaryEmbedding to extend the context of rotating embeddings, thus improving its performance in long context scenarios. Phi-3 uses a different prompt format than Llama. Specifically, the prompt for Phi-3 is structured as follows:
"""<|user|>
text
question
<|end|>
<|assistant|>."""
Unlike Llama, Phi-3 does not include a system section in the prompt. This choice may be due to the fact that Phi-3 was not trained using a prompt that included a system section. The absence of this section reflects a difference in the design and configuration of the training between the two models.

Minerva

Minerva is the first family of language models pre-trained from scratch for Italian, developed by Sapienza NLP in collaboration with FAIR and CINECA. Minerva models are open-access and bilingual (Italian and English), with about 50% of the pre-training data in Italian. The series includes three main variants

- Minerva-350M-base-v1.0: A compact, fast model suitable for tasks requiring quick responses and limited resources.
- Minerva-1B-base-v1.0: A balanced model with 1 billion parameters, ideal for more complex applications that require good performance without a large consumption of resources.
- Minerva-3B-base-v1.0: A powerful model with 3 billion parameters, optimised for advanced language solutions and high-quality text generation. Minerva represents a significant advancement for language models in Italian, offering smooth and accurate responses in bilingual contexts.

For Minerva I used the following prompt :
“’ <s>
phrase
text
question
Answer:
‘"’

astronomer/Llama-3-8B-Instruct-GPTQ-4-Bit

This HuggingFace repository contains a 4-bit quantised version of the Meta-Llama-3-8B-Instruct model. By reducing the memory footprint of the model to less than 6GB of VRAM (from the original 16.07GB). The 4-bit quantisation, achieved with the AutoGPTQ library, introduces minimal quality degradation compared to the original bfloat16 model, while significantly improving latency and throughput. The quantisation is calibrated with random samples from the wikitext dataset to ensure minimal loss of accuracy.

Results

These models were tested on the BBC News dataset. The use of this dataset is motivated by the fact that the text generation model is not used for its ability to create new text based on one's own knowledge, but rather for its ability to generate coherent and synthetic content from the documents provided. For this reason, a dataset suitable for summarisation is ideal. Furthermore, there is always a qualitative evaluation component in the responses produced by LLM models, such as the perception that the text was written by a human rather than a machine, which is difficult to measure in quantitative terms.
The Minerva model was considered for its particularity of having been developed in Italy, partly using Italian data. However, already from the first qualitative

evaluations, the results were not satisfactory: the model kept producing anomalous responses and inappropriate content. This could be due to an inadequately processed dataset or a lack of alignment. In fact, the model displayed discriminatory or prejudiced language and stereotypes. As a result of these issues, Minerva was discarded after the first evaluation.
In contrast, the LLama and Phi 3 models showed adequate responses under an initial qualitative evaluation, albeit with a clear perception of their artificial nature. Subsequently, both were tested on a summarisation dataset using metrics such as BERT-score and ROUGE. As shown in Table 4.2 for Italian and Table 4.1 for English, the results obtained were similar both semantically (BERT-score) and syntactically (ROUGE). However, the processing times were different: LLama, being a more complex model, was run on CPU due to hardware limitations, with higher response times than Phi 3, which was instead run on GPU, but also with considerable response times.

Considering the fact that the response times of both models were not acceptable for a chatbot and LLama's responses were qualitatively more satisfactory, these lead to the search for a quantized version. In fact, as we can see in Table 4.3, where the entire final pipeline was run with an MMR recovery for all three models, the quantized version of LLama maintained similar results to the non-quantized one, with the advantage of having very low response times, less than 5 seconds, compared to the other two models which had response times of over 20 seconds. For this reason, quantized LLama was chosen as the LLM model for text generation.

	PHI				LLAMA			
	Rouge-1	**Rouge-2**	**Rouge-L**	**BERTScore**	**Rouge-1**	**Rouge-2**	**Rouge-L**	**BERTScore**
Mean Precision:	0.50812	0.25053	0.48174	0.8941532	0.52427	0.24708	0.49311	0.8910702
Mean Recall:	0.34566	0.14917	0.32896	0.8703529	0.31861	0.13339	0.30015	0.865111
Macro AVG F1-Score	0.41143	0.187	0.39095	0.8820925	0.39635	0.17325	0.37316	0.8778987

Table 4.1: Results of the summarisation test with the English documents

	PHI				LLAMA			
	Rouge-1	**Rouge-2**	**Rouge-L**	**BERTScore**	**Rouge-1**	**Rouge-2**	**Rouge-L**	**BERTScore**
Mean Precision:	0.49935	0.2479	0.47233	0.7612496	0.54828	0.27569	0.50841	0.774398
Mean Recall:	0.36178	0.17742	0.3423	0.7359547	0.29974	0.13223	0.27797	0.7196477
Macro AVG F1-Score	0.41958	0.20682	0.39694	0.7483885	0.38759	0.17873	0.35943	0.7460196

Table 4.2: Results of the summarisation test with the Italian documents

	PHI	LLAMA	QUANTIZED LLAMA
Percentage of correct Answer	0.53	0.62	0.47
Percentage of correct or nearly correct answers	0.70	0.77	0.79
Mean time	24 s	79 s	4.67 s

Table 4.3: Results of the generation test with the final pipeline

4.4.2 Choosing the embedding model

Another key model in the RAG pipeline is the embedding model, used to store chunks in the vector database. Following the approach taken for LLMs, I selected only open source or open-weighted models available on Hugging Face. In particular, I examined the models that rank at the top of the MTEB leaderboard.
Before being tested on business documents, these models were evaluated on an Italian and an English document to obtain a general overview of their performance. I tested both models that showed excellent performance in English and one that stood out for excellent performance in languages other than English.

Alibaba-NLP/gte-large-en-v1.5

A first model tested was Alibaba-NLP/gte-large-en-v1.5, hereafter referred to as gte-large, which supports a context length of up to 8192 tokens and it is based on the BERT encoder architecture. It was trained exclusively on English data and has 434 million parameters. The model generates vectors with a dimension of 1024.

sentence-transformers/all-MiniLM-L6-v2

The sentence-transformers/all-MiniLM-L6-v2, hereafter referred to as Mini , model maps sentences and paragraphs in a vector space of size 384. It supports a maximum of 256 tokens and has been trained on over a billion data points. The model has 22.7 million parameters.

BAAI/bge-large-en-v1.5

The BAAI/bge-large-en-v1.5 model, hereafter referred to as Bge, generates 1024-dimensional dense vectors. It can process sequences with a maximum length of 512 tokens and uses 335 million parameters.

intfloat/multilingual-e5-large

The intfloat/multilingual-e5-large model, hereafter referred to as Multi, has 24 layers and produces embeddings with a size of 1024. It has 560 million parameters and was initialised from xlm-roberta-large, with subsequent training on a multilingual dataset of approximately 100 languages.

llama3/phi3

I evaluated Phi 3 and LLaMA 3 to generate embeddings to match their respective LLMs, but ultimately decided not to use them. These models are primarily designed for text generation, which means that their architecture and training are optimised

to produce smooth and consistent text, rather than to create efficient numerical representations. As a result, they require more computational resources, making the process slower and more expensive than models developed specifically for embedding.

Alibaba-NLP/gte-Qwen2-7B-instruct

The Alibaba-NLP/gte-Qwen2-7B-instruct, hereafter referred to as Qwen, is the largest of the embedding models tested, with a size of 7.6 billion parameters and an embedding size of 3584. It supports a maximum of 32,000 input tokens, thus offering considerable processing capacity. Moreover, it belongs to the same family as gte-large.

Results

Phi and LLaMA were initially tested, but it was quickly decided to switch to other models due to their nature. At the beginning of the tests, it became apparent that Qwen required a lot of processing time and could not be used with the GPU due to hardware limitations, also considering the use of other models in the pipeline. The other models were tested on one document in Italian and another in English, and their results are shown in Tables 4.5 and 4.4 respectively. In this case, we asked many questions for each document and I considered the times when the model retrieved the best chunk as the first document. The data show that the gte-large model performed very well in English compared to the other models. Although it provides acceptable results in Italian, the best in this language turns out to be the Multi model, justified by the fact that it was trained on a large set of languages. Therefore, for the subsequent experiments, these two models were chosen according to the language of the input data. With regard to the time taken by the models to return the chunk, all models achieved an acceptable result. So the time did not influence the evaluation much.

	gte-large	bge	mini	multi
Within 1st	**0.65**	0.6	0.35	0.55
Time	1.87	1.38	**1.098**	2.04

Table 4.4: Results of the test of the embedding models with the English document

	gte-large	bge	mini	multi
Within 1st	0.4	0.6	0.5	**0.7**
Time	1.71	1.4	**1.12**	1.99

Table 4.5: Results of the test of the embedding models with the Italian document

4.4.3 Choosing the vector db

A vector database stores, manages and indexes high-dimensional vector data. Data are stored as arrays of numbers called 'vectors'.
Vector databases are useful for three main reasons:

- Vector storage
  Vector databases store the outputs of an embedding model, the vector embeddings. They also store each vector's metadata—such as description and data type—which can be queried by using metadata filters. By storing these embeddings, the database enables quick retrieval during a similarity search, aligning the user's prompt with a corresponding similar vector embedding.

- Vector indexing
  Vectors need to be indexed to accelerate searches within high-dimensional data spaces. Vector databases create indexes on vector embeddings for search functions. The vector database indexes vectors by using an ML algorithm. Indexing maps the vectors to new data structures that enable faster similarity or distance searches between vectors.

- Similarity search based on querying or prompting
  Query vectors are vector representations of search queries. When a user queries the chatbot, the model computes an embedding of the query or prompt. The database then calculates distances between query vectors and vectors stored in the index to return similar results. Databases can measure the distance between vectors with various algorithms, such as nearest neighbor search. Measurements can also be based on various similarity metrics, such as cosine similarity.[2]

Two different vector databases were tested to see which one performed better.

Faiss

FAISS (Facebook AI Similarity Search) is an open-source library developed by Facebook AI Research for efficient similarity searching and clustering of dense vector embeddings. It offers a collection of algorithms and data structures optimised for different types of similarity search, enabling fast and accurate retrieval of nearest neighbours in high-dimensional spaces. Written in C++, FAISS provides complete wrappers for Python and implements some of the most useful algorithms on GPUs as well.[3, 4]

Qdrant

Qdrant is an open-source vector similarity search engine and database designed to efficiently handle high-dimensional data. It enables fast and accurate searching in large collections of vector embeddings, making it ideal for tasks based on semantic retrieval. It was chosen, in addition to Faiss, because, unlike the latter, it natively supports hybrid searching, combining semantic and keyword searching. [5]

Result

Although Qdrant offers some interesting features compared to Faiss, such as hybrid search (keyword-based search combined with semantic search), these could also be implemented later in a more understandable and manageable way. In addition, in practical use, no significant difference emerged between the two databases, so it was decided to continue with Faiss, already used in the initial tests, due to its familiarity and ease of use. This is therefore a purely personal choice.

4.4.4 Choosing the similarity distance

After testing and making a decision regarding the vector database, it remains to choose a metric to calculate the distance between two vectors, which will be used during the search phase. The available options include the similarity functions already mentioned in the section.

Euclidean distance

It is the most intuitive metric for measuring the minimum straight-line distance between two vectors in a multidimensional space. However, its disadvantage lies in the 'curse of dimensionality', as the computational complexity increases with increasing dimensions. The lower the value of the Euclidean distance, the greater the similarity between the vectors.[6]

$$d(\mathbf{a}, \mathbf{b}) = \sqrt{\sum_{i=1}^{n} (a_i - b_i)^2}$$

Where:

- $\mathbf{a} = (a_1, a_2, \ldots, a_n)$ is the first vector.
- $\mathbf{b} = (b_1, b_2, \ldots, b_n)$ is the second vector.
- n is the dimension of the vectors.
- $d(\mathbf{a}, \mathbf{b})$ represents the Euclidean distance between the vectors $\mathbf{a}$ and $\mathbf{b}$.

Inner product

This method is typically used to search for the maximum inner product in recommendation systems. The norm of the query vectors does not affect the ranking of the results. Similar to cosine similarity, given that the vectors are normalized.[7]

$$\text{MIP}(\mathbf{a}, \mathbf{b}) = \max_{\mathbf{b} \in D} \mathbf{a}^\top \mathbf{b}$$

Where:

- $\mathbf{a}$ is the query vector.
- $\mathbf{b} \in D$ represents the vectors in the database.
- $\mathbf{a}^\top \mathbf{b}$ is the dot product (or inner product) between the vectors $\mathbf{a}$ and $\mathbf{b}$.

Results

These two metrics were tested on company documents and the results can be summarised in Table 4.6. The Euclidean distance is the default metric for the Faiss database that was chosen to be used, while the maximum inner product is another metric that Faiss allows to be used. Cosine distance is also a possibility, but was not tested as it would have returned similar values to the inner product, given that vectors are normalized. As can be seen from the table, the results between the two methods show little difference, especially considering the first four documents returned. However, Faiss' default distance obtains better results, so this was chosen.

	MAXINNER	Euclidean(Faiss default)
within 1st	0.55	0.65
within 4th	0.7	0.75

Table 4.6: Results of the similarity test

4.4.5 Choosing the preprocessing phase

Document preprocessing is a fundamental step in Retrieval-Augmented Generation (RAG). There are several techniques that can be applied, depending on the type of documents and their characteristics. Not only does this process improve model performance, but it also ensures greater consistency and relevance in generative interactions.

Tokenization

One of the main preprocessing techniques in NLP is the tokenization and removal of stop words, i.e. the division of the text into units and the elimination of words that are not essential to identify the main topic. Although this technique is essential for simpler approaches, such as keyword search, it is useless and sometimes counterproductive for more complex methods, such as semantic search. For this reason, it was decided to avoid this type of processing.

Conversion into pdf

The documents used with this chatbot have different formats, mainly .doc, .docx and pdf, which requires different pre-processing with different libraries and leads to different text extraction methods. In order to resolve the differences in text formatting, especially for tables between Word and PDF documents, it was decided to convert all non-PDF documents into this format. Subsequently, several packages/-modules for extracting text from PDFs were tested, including PyPDF, PyMuPDF and PDFPlumber. All loaders showed good performance, but PDFPlumber proved to be the best at textual formatting of tables, thus reducing the need for further processing.

Recursive Character Text Splitting

Syntactic division is a technique in which documents, from which text is extracted, are divided on the basis of special characters defined in advance. The choice of separating characters depends on the language and grammatical structure to be analysed. It is also possible to set a minimum or maximum number of characters for each chunk, as well as the maximum number of overlapping characters between consecutive chunks.
In my case, the chosen separation characters were:

- '\n'
- '\n\n'
- '.'

- '`\uff0e`' (Fullwidth full stop)
- '`\u3002`' (Ideographic full stop)

I then tested two different sets of parameters:

- chunk size=2000, chunk overlap=200
- chunk size=400, chunk overlap=200

This technique is very simple and may be sufficient for documents of simple structure.

Semantic chunking

Semantic chunking is a technique that divides a text into blocks based on semantic meaning, rather than fixed rules such as the number of characters. It uses semantic embeddings, vector representations of text that capture meaning, to compare different parts of the text and group similar ones together. This approach allows blocks of text dealing with related concepts to be kept together, improving content coherence.
In the procedure, the text is divided into basic units (such as single sentences or windows of 3 consecutive sentences). For each unit, an embedding is generated. The embeddings are then compared to measure the semantic distance between consecutive sentences: the greater the distance, the more different the concepts are. When the difference between two embeddings exceeds a predefined threshold, a breakpoint is considered and the text is split into separate chunks. This makes it possible to separate portions of text dealing with different topics.
There are several methods for defining the breakpoint, the most common of which include :

- Percentile: The breakpoint is set to a predefined percentile, 85 in my case. If the difference between two embeddings exceeds this value, the text is split.
- Standard deviation: The threshold is based on the number of standard deviations from the mean of the differences. If a difference is greater than the set threshold, the division is carried out.

Translation and splitting

Since the LLM and embedding models used in this project were mainly trained on English data, it was decided to translate the texts of the documents to make them more similar to the training set. However, this led to a problem: in order to maintain an open source or otherwise free approach, the freely available translators

all have limitations on the number of characters that can be translated at once. Of the models tested, Google's translator was chosen because it offered the best ratio between quality and the maximum number of characters that could be translated, i.e. 5000.
Consequently, it was logical to apply the translation after splitting the texts into chunks, rather than before. Although the division by number of characters presents no problems, the semantic division created difficulties, as in the basic version there is no parameter for the maximum number of characters in the chunks. Changing the semantic division parameters affects the size of the chunks, but not enough to ensure that all of them remain under the 5000 character limit.
To solve this, it was decided to modify the syntactic splitting algorithm by introducing a parameter for the maximum length. In this way, when a chunk exceeds the set threshold, it is further subdivided, using either a new semantic division or a character division, thus respecting the 5000-character limit. This approach allowed the documents to be translated correctly, significantly improving performance.

Addition of file name and abbreviations and date processing

In an attempt to improve the performance of the chatbot, considering that it is intended for business use only, we tried to identify and exploit useful relationships between documents.

- Processing of dates: An improvement was found when responses contained dates expressed in the form 'dd name of the month yyyy' instead of the format 'dd/mm/yyyy'. Consequently, it was decided to expand all dates in this format within the chunks.
- Abbreviations: Given the frequent use of abbreviations in documents, a set of abbreviations was created with the relevant explicit words, which were replaced during preprocessing.
- File names: Since document names are very descriptive, clearly indicating the content, a preprocessing was applied that pastes the file name at the end of each chunk. This approach helps to retrieve the correct chunk even when there are similar paragraphs in different documents.

Footer removal and text merging

Following the same principle as previously applied, most documents had a footer at the end of each page containing company information and the like. It was therefore decided to remove these footers using regular expressions. This made it possible both to reduce the size of the chunk dataset and to improve semantic division, as

footers often caused premature division of text blocks, interrupting segments that dealt with the same topic.
The preprocessing process involved removing footers and joining consecutive chunks, based on punctuation parameters. If a block preceding a footer ended without punctuation or with a comma, it was merged with the next one. Conversely, if a block ended with a full stop or a large space, it was not merged with the next chunk.

Results

The preprocessing phase is a crucial part of the pipeline, as good data preparation can significantly improve results. For this reason, much attention was paid to this phase, testing different strategies for dividing documents into chunks and various text processing techniques.

Starting with the simplest method, i.e. the division of documents by number of characters, two sets of parameters were tested on business documents in Italian, using the modified MRR (Mean Reciprocal Rank) metric. The results of these tests, together with those of the semantic division technique, are summarised in the table 4.7, where a simple search based on the MMR was employed. With regard to character division, it was observed that a larger number of characters improved the results, probably due to the increased context. Semantic division without size constraints proved to be slightly better, following the same logic. Therefore, I focused further on this methodology and tried to refine it.

Considering that LLM models perform better in English, it was decided to translate the chunks into English, imposing a limit of 5000 characters. To respect this constraint, the semantic division was modified in two ways: either with an iterative semantic division reducing the percentile by 15 points until the limit was respected, or with a character-based division with a chunk size of 2000. Both options were tested, and the results, visible in the table 4.8, were obtained using a search with MultiQuery and a subsequent reranking. Again, the differences between the methods are small, but character division showed better results. Furthermore, this approach is more deterministic and less time-consuming, as semantic division requires an uncertain number of iterations to reach the 5000-character limit.

After optimising the division technique, I focused on text processing. In addition to the initial conversion of all documents into PDF format, the first pre-processing phase involved the transformation of abbreviations and dates, and the addition of the document name at the end of each chunk. This phase led to a clear improvement, with satisfactory results for the creation of a chatbot based on open source templates, as shown in the 4.9 table, where several preprocessing processes were applied with a hybrid MultiQuery and BM25 search followed by reranking.

Subsequently, to address the problem of footers on most pages, a further preprocessing step was introduced to remove footers and merge consecutive pages that dealt with similar topics. Contrary to expectations, as can be seen in the table, this preprocessing worsened the results compared to the first phase, with even lower performance than the simple semantic division.

Consequently, it was decided to adopt semantic division as the preprocessing pipeline, with the possibility of resorting to character division if the 5000-character limit was exceeded. The text processing of these chunks includes the transformation of abbreviations and dates, as well as the addition of the document name, previously converted to PDF.

	Character with size of 400	Character with size of 2000	Semantic
Within 1st	0.3	0.4	0.45
Within 4th	0.35	0.45	0.5

Table 4.7: Results of semantic and character level chunking comparison using MMR.

	Sem(rec)	Sem(sem)
Within 1st	0.6	0.55
Within 4th	0.75	0.7

Table 4.8: Results of comparison between semantic division with 5000 character limit using a subsequent semantic division and at character level, using a search with Multiquery and reranking.

	Sem(rec)	Sem(rec)+preprocessing	Sem(rec)+preprocessing+foooter
Within 1st	0.6	0.71	0.31
Within 4th	0.75	0.94	0.51

Table 4.9: Results of the comparison of different text processing, using a search with MultiQuery and reranking.

4.4.6 Choosing the reranker

Reranking is a process of re-evaluating and reordering retrieved documents according to their relevance to a query, with the aim of improving the accuracy and quality of the results. This technique is particularly important in advanced language processing, where it allows optimising information retrieval and aligning answers to user or domain-specific needs.
Reranking is usually implemented in two-phase retrieval systems:

- First phase (retriever): An embedding model, or bi-encoder, quickly retrieves a set of relevant documents from a large dataset. This model is very fast because it pre-calculates the document vectors before the query, compressing all possible interpretations of the document into a single vector. When a query arrives, only the query vector is computed, which is compared with the document vectors via metrics such as cosine similarity. However, this process can lead to information loss, as the model cannot adapt the meaning of the document to the specific context of the query.

- Second step (reranker): A reranker model, or cross-encoder, reorganises these documents based on a deeper and more precise analysis. It examines each query-document pair in real time, calculating a similarity score to establish actual relevance. This model, performs new processing during the query, analysing query and document together. This avoids information compression, making the result more accurate, but at the expense of speed, as each query-document pair requires full computation.

Thus, bi-encoders are fast but less accurate, because the compression of data into vectors results in a loss of important details. In contrast, cross-encoders provide much more accurate results, as they evaluate documents according to the specific query, but this process is slower.[8]

Without reranking

In the initial phase of the pipeline, a methodology without reranking was adopted, using a baseline without this component so that the performance of the pipeline could then be evaluated once the reranking model was added.

BAAI/bge-reranker-base

This reranking model, with 278 million parameters, supports both Chinese and English.

BAAI/bge-reranker-large

With 560 million parameters, this reranking model is a larger and more powerful version of its predecessor, also capable of handling Chinese and English.

With langchain and wothout langchain

LangChain is a library widely used with LLM and NLP, as it simplifies the implementation of various tasks.
The testing of the reranking task with the two previously described models was performed in two modes: one using the LangChain library and the other applying the model directly. Although both modes produced correct results, the application without the use of LangChain proved to be more effective, probably due to the library's internal mechanisms. However, in terms of execution time, the two solutions were similar. [9]

Results

Table 4.10 summarises the results of the three options tested: without reranking, with the bge base model and with the bge large model, indicating the percentage of times the correct chunk is returned as first or within the first 4 results.

As one would expect, reranking improves performance compared to no reranking. The results of the two models, bge base and bge large, are very similar, but bge large was chosen because of its slight advantage over bge base. Furthermore, although bge large is larger in size, both models remain small enough to be easily run on GPUs.

	No Rerank	Bge Base	Bge Large
Within 1st	0.5	0.6	0.65
Within 4th	0.65	0.75	0.75

Table 4.10: Results of the rerank model comparison.

4.4.7 Keyword and Hybrid Search

Keyword-based search, also known as sparse vector search, relies on generating sparse embeddings where most values in the vector are zero except for a few non-zero values. These sparse embeddings are created using algorithms such as explained in 2.2.1 In a keyword-based search the main phases are:

- Preprocessing: The search query and documents are preprocessed to extract relevant keywords and remove noise.

- Next, the other steps are identical to the semantic search, but with a different embedding algorithm.

Sparse vectors are particularly effective in domains and scenarios where there are many rare keywords or specialised terms. Therefore, in contexts such as this chatbot with technical words, both in the technology and business fields, they are very useful. Furthermore, this methodology is designed to be applied especially to temporal searches, as an algorithm based on keywords is more likely to correctly identify a paragraph containing a specific date than a model based on text meaning.[10, 11]

The idea is to combine these two methodologies, as they address the search problem in different ways, thus allowing each to compensate for the weaknesses of the other.

Initially, sparse vector search alone is tested and, once good results are obtained, both techniques are integrated, thus creating a hybrid search approach.

Results

The idea of also using a keyword search was considered from the outset. As can be seen in the 4.11 table, a test was conducted with character division using the MRR metric. However, the results were disappointing, as the keyword search did not add any significant information compared to the semantic search. The same result was obtained with semantic division without preprocessing.

With the integration of semantic division and preprocessing, as shown in the table, the combination of the two search modes produced a significant increase compared to the exclusive use of the MMR. Consequently, it was decided to adopt a hybrid search.

Parameters	**metric**	**MMR**	**BM25**	**MMR+BM25**
Character chunking with size of 2000	within 1st	0.4	0.15	0.40
	within 4th	0.45	0.25	0.45
Sem(rec) without preprocessing	within 1st	0.55	0.2	0.55
	within 4th	0.65	0.4	0.65
Sem(rec) with preprocessing	within 1st	0.55	-	0.65
	within 4th	0.76	-	0.88

Table 4.11: Results of the comparison between semantic, kewword-based and hybrid search.

4.4.8 Searching with MMR

The similarity search returns the answers closest to your question. However, to provide the model with more complete information, it may be useful not to focus exclusively on the most similar texts.

The idea behind using MMR (Maximum Marginal Relevance), as employed in text summarization, is to reduce redundancy and increase diversity in the results.[12, 13]
The operation is quite simple:

- First, we use similarity search to obtain the most similar documents to the query, denoted by fetch_k.
- Next, we select from these the k most diverse documents.

This can be summarised with the following formula:

$$\text{MMR} = \arg \max_{D_i \in R \setminus S} \left[\lambda \cdot \text{sim}(D_i, Q) - (1 - \lambda) \cdot \max_{D_j \in S} \text{sim}(D_i, D_j) \right] \tag{4.1}$$

where:

- D_i is a candidate document.
- R is the set of documents retrieved.
- S is the set of selected documents.
- $\text{sim}(D_i, Q)$ is the similarity between the document D_i and the query Q.
- $\text{sim}(D_i, D_j)$ is the similarity between documents D_i and D_j.
- λ is a balancing parameter between relevance and diversity.

Results

Although this technique is among the simplest, as shown in Table 4.12, it performed more than satisfactorily both at the beginning of the tests, using only the semantic chunker, and in the final phase, combined with the hybrid search. Compared with more complex techniques such as MultiQuery and Hyde, although it performed less well, it still maintained good results, distinguished above all by a significantly shorter average execution time than the other two methods.

Considering therefore the good performance and short response times, which are ideal for the creation of a chatbot, the exclusive use of MMR was chosen as the final solution for the chatbot.

Pre-processing	Metric	MMR	HyDE	MultiQuery
Semantic Chunker	Within 1st	0.45	0.4	0.3
	Within 4th	0.5	0.45	0.4
Sem(Rec) BM25	Within 1st	0.65	0.71	0.71
	Within 4th	0.88	0.91	0.94
	Mean time	0.67 s	20.7 s	50.23 s

Table 4.12: Results of the comparison between MMR, MultiQuery and HyDE

4.4.9 Searching with Hyde

Hypothetical Document Embeddings(HyDE), instead of focusing on the similarity between query embeddings, focuses on answer embeddings. In practice, it generates a 'hypothetical' answer with the help of an LLM model (such as GPT-3) and then looks for a match in the document embeddings.
However, this approach has a limitation: it does not always guarantee optimal results. For instance, if the subject matter is completely unknown to the language model, the effectiveness of the method decreases, increasing the risk of producing incorrect information.
A pre-trained LLM on specific documents would be an ideal component for the application of this method. In my case, however, as I had neither a large test dataset nor the hardware resources to fine-tune a model, the task was tested using the same LLM used to generate the answers. This may seem counterintuitive, but the ideas behind the test are that:

- in some cases, the questions asked by the users to the chatbot might be rather general or not too specific. An LLM trained on a wide variety of data could then provide an answer similar to the correct one, incorporating details that the research phase could refine.

- that drove the test is related to the generation of words that differ from the initial question, even if not completely accurate, but still remain relevant to the topic of the question. This could lead to a broader and more diverse search, a principle on which the MultiQuery method, described in section 4.4.10, is based.

Results

Since this method was not fully implemented, as no finetuning was performed, the results were nevertheless more than decent, as shown in Table 4.12, where it is compared with MMR and MultiQuery.

Hyde performed very well, even matching MultiQuery in terms of performance. However, with an average execution time of 20 seconds in the final pipeline, it is not an optimal choice for the needs of this project.

4.4.10 Searching with MultiQuery

In the application of a simple search, the pipeline may only return documents that contain exactly the words in the query. This can become problematic when the documents and query use synonyms for the same concept or similar terms for different concepts. In these cases, the use of a query transformation, similar to the one implemented by HyDE, can be useful. The Multiquery method generates several variations of the original query, allowing more facets of the user's intent to be captured. This broadens the search and makes it possible to retrieve documents that may not contain exactly the keywords, but still offer relevant information.[14]
To implement this approach, an LLM model is used to generate new query variants from the initial query provided by the user.[15]
In this project, the same model was used to generate the query text, but with a different prompt:

"""<|user|>You are an AI assistant. Your task is to generate 5 different versions of the user's provided question to retrieve relevant documents from a vector database. By generating multiple perspectives on the user's question, your goal is to help the user overcome some of the limitations of similarity-based search. Provide these 5 alternative questions separated by new lines. Original question: question<|end|><|assistant|>"""

To get the most out of this method, we performed the tests by also varying the main parameter, i.e. the number of new questions generated. Furthermore, it was possible not to include the original question, using only the derived questions. However, this option was quickly discarded as, despite the quality of the LLM models used, the derived questions might not accurately reflect the user's intent.

Results

This is one of the most complex techniques within RAG, and as expected, it performed best in terms of performance, as shown in the table 4.12, where it is compared to Hyde and MMR.

Although it provided the best performance in information retrieval, reaching almost the maximum for retrieval of the correct chunk within the first 4 results with the final preprocessing pipeline, the complexity of this method makes it slow, with an average execution time of 50 seconds. This slowdown is partly due to a

poor hardware configuration. With better hardware, however, this technique could become the ideal choice for a chatbot, combining excellent performance with more adequate response times.

4.4.11 ParentRetriever

The choice of chunk size when splitting documents at character level involves balancing a trade-off: shorter chunks make embeddings more accurate, while longer chunks allow the full context of the document to be maintained, preventing meaning from being lost between chunks.
The Parent Document Retriever addresses this problem by dividing documents into smaller chunks and storing them in the leaves of a tree structure. The top nodes contain larger portions of text that group the smaller chunks together. During retrieval, information is searched in the smaller chunks, but the system returns the larger documents to preserve context and improve search efficiency.[16, 17]

Results

This technique, although among the most complex and successfully used in high-performance RAG pipelines, did not yield the same results in this project, especially when compared to the preprocessing described above applied to documents. Since it did not produce the expected results, it was decided to maintain the already established pipeline.

4.4.12 Prompt selection

With the introduction of the new LLMs (Large Language Models), a new area has emerged within machine learning called 'Prompt Engineering', the aim of which is to optimise the prompts used with these models to improve responses and maximise performance. This highlights one of the main challenges of LLMs: even a single modified word in the prompt can lead to very different results.
In this project, we therefore experimented with various prompts, also applying advanced techniques. We started with a generic prompt:
"You are a helpful assistant that extracts informations from a given text.
Only answer to the given question with the information you can find here:"

few-shot, one-shot and zero-shot

As described in section 2.5, some Prompt Engineering techniques involve the addition of zero, one or more examples within the prompt to direct the model towards a predefined response. Typically, no examples are added in this type of

prompt, but we conducted a test by including one. This is because the models are already trained to answer questions from users.

200 dollars version prompt

One technique that highlights the indeterminacy of neural network-based models is the addition of a reward or penalty. In this case, the insertion of a $200 'tip' to incentivise a more accurate response was tested.[18]
This solution was explored because, with the use of a simpler prompt, not entirely correct or incomplete answers were obtained, even though the right chunk was selected.
The prompt used was as follows:

"You are a helpful assistant that extracts informations from a given text.
I will give you 200 $ if you respond in a detailed manner and verify that the answer contains all the information requested by the question without adding anything unnecessary.
Follow these steps:
Carefully read the question.
Carefully read the answer.
CHECK THAT THE ANSWER CONTAINS ALL THE INFORMATION REQUESTES BY THE QUESTION AND NOTHING MORE.
Ensure that the answer does not contain any additional, unnecessary information and that the information in the answer is not contradictory.
Only answer to the given question with the information you can find here:"

Results

Since the choice of prompt is not entirely deterministic, it is difficult to determine which is the best option.

However, using the '$200 tip' prompt and specifying to the model, in capital letters, to double-check, gave more accurate results and more complete answers. Therefore, it was decided to adopt this prompt, without resorting to one-shot or few-shot methods.

4.4.13 Choice of Generation

The RAG pipeline can be divided into two main components: the retrieval part and the generation part. While the retrieval phase is crucial and requires more attention to optimise the process, the generation phase offers less room for variation, as it has fewer models than retrieval and fewer techniques that can be applied.

Compression

The most common technique used in this task is Compression, which is particularly useful when the chosen model has a limited token window or when the available hardware resources are not particularly powerful.
In this process, once the documents relevant to the query have been retrieved, they are processed by a compressor that selects only the parts most relevant to the query, thus reducing the overall size. This approach can lead to a significant reduction in data, especially in the case of very large documents, where only a few lines contain the necessary information.

Results

The compression technique and its application also depend on the choice of model and the number of tokens that can be handled. As shown in Table 4.13, a comparison is presented between the phi 3 and llama 3 quantized models, using 2 or 4 documents. For llama, it was not necessary to test the option with 2 documents, since with 4 documents compression was not necessary, and a larger number was not tested to avoid confusing the LLM model.

In the case of phi 3, having a smaller input window, compression was necessary when using 4 documents. This configuration performed better than quantized llama, confirming the effectiveness of compression and the benefits it can bring to RAG.

However, as quantized llama was chosen for its significantly faster response times than phi 3 (almost five times faster, as can be seen in the table), it was decided not to use compression, as it is not necessary for quantized llama .

Model	**metric**	**2 Doc**	**4 Doc**
Phi	compression	no	yes
	correct	0.41	0.53
	correct or nearly correct	0.70	0.70
	mean time	17 s	24 s
Quantized Llama	compression	-	no
	correct	-	0.47
	correct or nearly correct	-	0.79
	mean time	-	4.67 s

Table 4.13: Results on compression

4.5 Text-to-sql

Unlike language models for generic text generation, for which there are many good open source alternatives and new ones are constantly emerging, in the text-to-SQL task there are not many performing models. In fact, there is no main model that stands out from the others. With this in mind, the research focused mainly on Hugging Face. Two models were selected that show good performance in commonly used tests for this task, with the former being much better known and recommended than the latter.

4.5.1 Sql-coder

The defog/sqlcoder-7b-2 model is a text-to-SQL model developed by Defog, Inc, with 6.74 billion parameters. It was refined from CodeLlama-7B, a model based on llama 2. This model was evaluated using SQL-Eval, a PostgreSQL-based evaluation framework created by Defog to test and align the model's capabilities. However, a significant limitation is that the model was only trained on PostgreSQL, which could affect its performance with other SQL databases.[19]

Application

According to the model's Hugging Face page, there is a specific prompt on which the model has been trained. The prompt requires the relational schema to be provided and the question to be expressed twice: once at the beginning and once at the end of the prompt.
Below is the prompt used:

```
### Task
Generate a SQL query to answer [QUESTION]{user_question}[/QUESTION]

### Database Schema
The query will run on a database with the following schema:
{table_metadata_string_DDL_statements}

### Answer
Given the database schema, here is the SQL query that [QUESTION]
{user_question}[/QUESTION]
[SQL]
```

Results

In order to test the performance of this model, a test database was used on which a number of queries were carried out with two different prompts, which differed in some final sentences.

The variation of prompts produced different results, as shown in Table 4.14, thus confirming the importance and risks associated with the choice of prompt for these models. However, since the model was trained exclusively on PostgreSQL queries and the decision was made to use a SQLite database, it was not possible to change the query type except by fine-tuning, which cannot be done due to the lack of hardware and data.

-	Prompt 1	Prompt 2
percentage of correct queries	0.65	0.5

Table 4.14: Test results with the Sql-coder model

4.5.2 PipableAI/pip-library-etl-1.3b

The PipableAI/pip-library-etl-1.3b model has 1.35 billion parameters and was developed using advanced techniques such as softmax cross entropy, a modified version of policy gradient and Q loss, optimised in an Expectation-Maximisation (EM) context. Thanks to these methodologies, the performance for the mentioned tasks is comparable to that of much larger language models, such as GPT-3.5, thus demonstrating considerable efficiency and capacity despite the model's smaller size.[20]

Implementation

The implementation of this model to our task was rather simple, as no complex preprocessing had to be implemented. The retrieval of the data to be supplied to the model consists of the required query and the relational database schema of the tables containing the information. This can be done a priori or by retrieving metadata and/or tables depending on the database used.
Since the model is trained on a large amount of data, consisting of schema and query pairs, the prompt used to query the model is specific and provided by the Hugging Face page. The basic prompt is as follows:

“””<schema>schema with cols described</schema>
<question>Write a sql query to ....</question>
<sql>"""

From this prompt, phrases can be added at the beginning to address the model according to one's needs. However, as with other LLMs, this option should be treated with caution, as it could lead to unexpected results.
With this in mind, two slightly different prompts were created for the two chatbot tasks: one for creating the table and one for generating the graph.
For generating the tables:
"""Generate a simple SQL query from the schema mentioned for the following question.
If you don't find the a possible query, return SELECT 'cinoooo' AS response;
<schema>schema</schema>
<question>question</question>
<sql>"""

For generating the charts:

"""Generate a simple SQL query from the schema mentioned for the following question.
If you don't find the a possible query, return SELECT 'cinoooo' AS response;
If there are only 2 column, then you must put the more important one as second one. <schema>schema</schema>
<question>question</question>
<sql>"""

The version of the graph prompt is quite limiting, as the aim was to demonstrate the possibility of creating graphs, focusing on simple types such as lines and bars, using only two sets of data.

Results

To evaluate this model, it was tested on a test SQLite database and a few queries were made.

As shown in table 4.15, the length of the returned tables was correct in all cases, but the number of columns was greater than expected in the correct query. However, this did not result in a significant error. Furthermore, although the average response time is high, it is still reasonable for use in this type of software. Therefore, it was decided to adopt this model as the final solution for producing search queries.

Equal output to less than one distinction	Equal output length	Query generation time
50%	100%	8.032 s

Table 4.15: Test results with the PipableAI/pip-library-etl-1.3b model

4.5.3 Vanna ai

Vanna ai is a Python package designed to use retrieval augmentation to generate accurate SQL queries for your database by exploiting large language models (LLM). The operation of Vanna ai is simple and consists of two basic steps: first you train a RAG model on the specific data, and then you can pose queries to get the SQL queries ready to execute.
The vn.train(...) method allows you to train the system, enriching the reference corpus with various types of information, such as DDL statements to understand table structure, documentation strings concerning the database or industry, and SQL queries frequently used in your organisation. In addition, pairs of SQL questions and queries can be provided, which helps the system better interpret the context of questions, especially in ambiguous situations.
After training, the function vn.ask(...) allows you to ask questions to the system, which uses the reference corpus to generate the required SQL queries. Before you can start making queries, it is essential that the system has adequate training data available. With Vanna ai, the process of interacting with databases can be optimised and simplified, making the generation of SQL queries more accessible and intuitive.[21]

Implementation

Vanna ai provides already written code for some vector databases, LLM and SQL databases. However, most of these models and databases are chargeable and do not align with the goal of this project. Therefore, all methods were implemented by overwriting them in order to ensure correct operation with the models already in the pipeline. Training was carried out using the relational schema, enriching it with metadata to improve the retrieval process.

Results

To test this framework, the same database was used to evaluate the pipable model chosen for query creation, and several queries were made. In all cases, the framework managed to return the part of the relational schema required for the queries.

This represents a considerable step forward, since the chosen model has a limited input window and, for large databases, it is not possible to provide the entire relational schema.

Therefore, especially in connection with the application on the company database, it was decided to implement this pipeline.

Chapter 5

Conclusion

The aim of this project was to develop a chatbot based on machine learning models capable of answering users' questions naturally and accurately, using data from company documents or databases, in order to improve productivity. The entire pipeline involves a document preparation phase, which is divided into blocks and transformed into vectors using an embedding model, then stored in a vector database. When the user makes a request, the system searches for the most relevant documents using a similarity search between vectors. The retrieved documents are reordered using a reranking model, based on similarity to the query. The first four selected documents are then passed on to the text generation model, which uses the contained information to appropriately answer the user's query.
In terms of interaction with SQL databases, each table is stored separately in a vector database, along with any previous queries and their metadata. When the user poses a query, a process similar to that for documents is applied to the database tables, retrieving the relational schemas needed to generate the SQL query via a machine learning model. The generated query is executed and returned in table or graph form, depending on the user's needs.

Being designed to assist users in various tasks, the two key parameters to consider are accuracy and speed of response. An optimal response time should be 2-3 seconds, but values of up to 5 seconds are still acceptable. The main objective was to achieve the highest possible accuracy. In order to achieve a good balance between accuracy and speed, the MMR (Maximal Marginal Relevance) algorithm was used, resulting in an average of 4.67 seconds per response and an accuracy of 88% in the correct documents selected within the first four. The responses generated by the language model were considered correct or nearly correct in 79% of the cases. These results show that the time and accuracy targets were met, taking into account the limitations imposed by the company's hardware and the open source choices made.

There are companies and frameworks that allow the creation of similar chatbots, but they often use proprietary models with high performance, such as those of the GPT family, and make extensive use of cloud services for managing documents and hardware resources, achieving better results thanks to scalable resources. The solution proposed in this project differs mainly in its cost-effective approach, as it relies on open source or open-weighted machine learning models using local enterprise hardware. This approach reduces costs but imposes some limitations in terms of performance, while still achieving good value for money.

The main challenge faced during the development of the project was the limited hardware, in particular the inability to run the models entirely on the corporate GPUs due to the size of the weights, which exceeded the capacity of the two available 16 GB GPUs. It was therefore crucial to find models that offered the right balance between size and performance. Consequently, special attention was paid to document preprocessing, as this stage was less dependent on the GPU. Here again, the search for optimal preprocessing techniques was complex but essential to extract the maximum from the data.

Despite the results achieved, some limitations remain unresolved. For instance, the system struggles to answer queries that require information distributed over numerous documents or to handle the ever-increasing volume of data, as this would result in longer search times. However, these limitations do not jeopardise the use of the chatbot for the specific business target group, consisting mainly of small and medium-sized enterprises, which could still benefit from the system, even with the present limitations.

Despite the technical challenges, the chatbot is well suited to the business use for which it was conceived, especially in contexts of collaboration with business partners or other companies in the sector. The solution proves ideal for small and medium-sized companies, which benefit from a cost-effective and high-performance system. Technical limitations are not a significant obstacle in these cases, given the scale of use.

As the project is still in the development phase, there are several future perspectives that could improve the user experience, such as the implementation of context retention between answers provided by the chatbot. This would improve the model's ability to respond to subsequent requests for clarification, but has not been implemented as the focus has been on the accuracy of a single answer per question. Another important step will be to improve the security and privacy of corporate data by controlling access to documents based on user roles and authorisations.

The integration of a RAG (Retrieval-Augmented Generation) model is essential to reduce the 'hallucinations' generated by LLM (Large Language Models) and ensure answers based on real document sources. Security and privacy management, especially in document access, will be central to the final phase of the project. These aspects are crucial not only to ensure the integrity of responses, but also to address the ethical issues related to the use of artificial intelligence-based chatbots.

The project has demonstrated its potential in improving business productivity and could be successfully applied in other enterprises. Machine learning technologies, and in particular neural networks, are becoming key tools for companies that want to remain competitive. This chatbot represents a concrete example of how artificial intelligence can be used to optimise processes and increase staff efficiency, confirming the importance of such technologies in the modern workplace.

Bibliography

[1] *BBC News Summary*. www.kaggle.com. URL: https://www.kaggle.com/datasets/pariza/bbc-news-summary (cit. on p. 28).

[2] *What is an vector database? | IBM*. www.ibm.com. URL: https://www.ibm.com/topics/vector-database (cit. on p. 39).

[3] Lalithkumar Prakashchand. *Similarity Search with FAISS: A Practical Guide to Efficient Indexing and Retrieval*. Medium, June 2024. URL: https://medium.com/@devbytes/similarity-search-with-faiss-a-practical-guide-to-efficient-indexing-and-retrieval-e99dd0e55e8c (cit. on p. 39).

[4] Ajithkumar M. *Working with FAISS for Similarity Search - Ajithkumar M - Medium*. Medium, Nov. 2023. URL: https://iamajithkumar.medium.com/working-with-faiss-for-similarity-search-59b197690f6c (visited on 10/06/2024) (cit. on p. 39).

[5] Vyom Modi. *Building a High-Performance RAG-Based AI with Qdrant, Groq, LangChain, and DAGWorks Hamilton*. Medium, May 2024. URL: https://vyom-modi.medium.com/building-a-high-performance-rag-based-ai-with-qdrant-groq-langchain-and-dagworks-hamilton-fb1baa7415bc (visited on 10/06/2024) (cit. on p. 40).

[6] Stepkurniawan. *Comparing RAG Part 3: Distance Metrics; (Similarity Index) in Vector Stores*. Medium, Jan. 2024. URL: https://medium.com/@stepkurniawan/comparing-similarity-searches-distance-metrics-in-vector-stores-rag-model-f0b3f7532d6f (visited on 10/06/2024) (cit. on p. 40).

[7] facebookresearch. *MetricType and distances*. GitHub, Jan. 2023. URL: https://github.com/facebookresearch/faiss/wiki/MetricType-and-distances (visited on 10/06/2024) (cit. on p. 41).

[8] *Rerankers and Two-Stage Retrieval | Pinecone*. www.pinecone.io. URL: https://www.pinecone.io/learn/series/rag/rerankers/ (cit. on p. 47).

[9] MyScale. *Enhancing Advanced RAG Systems Using Reranking with LangChain*. Medium, June 2024. URL: https://medium.com/@myscale/enhancing-advanced-rag-systems-using-reranking-with-langchain-523a0b840311 (visited on 10/06/2024) (cit. on p. 48).

[10] Csakash. *Hybrid Search a method to Optimize RAG implementation*. Medium, Feb. 2024. URL: https://medium.com/@csakash03/hybrid-search-is-a-method-to-optimize-rag-implementation-98d9d0911341 (visited on 10/06/2024) (cit. on p. 49).

[11] Nirant Kasliwal. *What is a Sparse Vector? How to Achieve Vector-based Hybrid Search - Qdrant*. Qdrant.tech, 2023. URL: https://qdrant.tech/articles/sparse-vectors/ (visited on 10/06/2024) (cit. on p. 49).

[12] Aditya Kumar. *Maximal Marginal Relevance to Re-rank results in Unsupervised KeyPhrase Extraction*. Medium, Oct. 2019. URL: https://medium.com/tech-that-works/maximal-marginal-relevance-to-rerank-results-in-unsupervised-keyphrase-extraction-22d95015c7c5 (visited on 10/06/2024) (cit. on p. 50).

[13] Mariya Mansurova. *RAG: How to Talk to Your Data - Towards Data Science*. Medium, Nov. 2023. URL: https://towardsdatascience.com/rag-how-to-talk-to-your-data-eaf5469b83b0 (visited on 10/06/2024) (cit. on p. 50).

[14] Plaban Nayak. *Advanced RAG — Improving retrieval using Hypothetical Document Embeddings(HyDE)*. Medium, Nov. 2023. URL: https://medium.aiplanet.com/advanced-rag-improving-retrieval-using-hypothetical-document-embeddings-hyde-1421a8ec075a (visited on 10/06/2024) (cit. on p. 52).

[15] *Multi-Query - Advanced RAG Techniques: Choosing the Right Approach*. Educative, 2015. URL: https://www.educative.io/courses/advanced-rag-techniques-choosing-the-right-approach/multi-query (visited on 10/06/2024) (cit. on p. 52).

[16] *Parent Document Retrieval (PDR) - Advanced RAG Techniques: Choosing the Right Approach*. Educative, 2015. URL: https://www.educative.io/courses/advanced-rag-techniques-choosing-the-right-approach/parent-document-retrieval-pdr (visited on 10/06/2024) (cit. on p. 53).

[17] *How to use the Parent Document Retriever | LangChain*. Langchain.com, 2024. URL: https://python.langchain.com/docs/how_to/parent_document_retriever/ (visited on 10/06/2024) (cit. on p. 53).

[18] *Does Offering ChatGPT a Tip Cause it to Generate Better Text? An Analysis*. minimaxir.com, Feb. 2024. URL: https://minimaxir.com/2024/02/chatgpt-tips-analysis/ (cit. on p. 54).

[19] *defog/sqlcoder-7b-2 · Hugging Face*. Huggingface.co, 2024. URL: https://huggingface.co/defog/sqlcoder-7b-2 (visited on 10/06/2024) (cit. on p. 56).

[20] *PipableAI/pip-library-etl-1.3b · Hugging Face*. Huggingface.co, 2024. URL: https://huggingface.co/PipableAI/pip-library-etl-1.3b (visited on 10/06/2024) (cit. on p. 57).

[21] *Vanna.AI Documentation*. vanna.ai. URL: https://vanna.ai/docs/ (cit. on p. 59).