UNIVERSIDAD PONTIFICIA COMILLAS

# WIKI LLM: DESIGNING A RELIABLE ASSISTANT CHATBOT FOR EFFICIENT QUERIES IN A CORPORATE WIKI

TRABAJO FIN DE MÁSTER

AUTHOR

GUILLAUME GUERS

MADRID, JUNE 2025

# Declaration of Authorship

**I declare, under my responsibility, that the project presented with the title**

**WIKI LLM: DESIGNING A RELIABLE ASSISTANT CHATBOT FOR EFFICIENT QUERIES IN A CORPORATE WIKI**

at the ETS of Engineering - ICAI of Universidad Pontificia Comillas for the academic year 2024/25

is of my authorship, original, and unpublished, and has not been presented previously for any other purpose.

The project is not plagiarized from another, neither in whole nor in part, and the information taken from other documents is properly referenced.

Signed: Guillaume Guers Date: 02 / 06 / 2025

GUERS

Authorized for submission of the project

**THE PROJECT SUPERVISOR**

Antonio González

Signed: Antonio González Suárez Date: 02 / 06 / 2025

GONZALEZ SUAREZ ANTONIO - 71901888R

Firmado digitalmente por GONZALEZ SUAREZ ANTONIO - 71901888R
Fecha: 2025.06.04 15:07:30 +02'00'

UNIVERSIDAD PONTIFICIA COMILLAS

# WIKI LLM: DESIGNING A RELIABLE ASSISTANT CHATBOT FOR EFFICIENT QUERIES IN A CORPORATE WIKI

TRABAJO FIN DE MÁSTER

AUTHOR

GUILLAUME GUERS

TUTOR

ANTONIO GONZÁLEZ SUÁREZ

COLLABORATIVE ENTITY

BAOBAB SOLUCIONES

# Acknowledgments

I would like to begin by expressing my heartfelt gratitude to Baobab Soluciones and its manager, Rafael Salafranca, for giving me the opportunity to carry out this project in such a stimulating and supportive environment. From the very beginning, they showed genuine concern for my well-being and entrusted me with a master's thesis that aligned perfectly with my interests. Their friendliness and continued support throughout the process gave me the time, confidence, and freedom to fully develop and explore this work.

A special thanks goes to Antonio González, my tutor and project supervisor. His constant involvement, approachability, and unwavering dedication have been invaluable. I am truly grateful for the time and effort he invested in guiding me and helping shape the project into what it has become.

I would also like to thank the Universidad Pontificia Comillas, and in particular Carlos Morrás, director of the Master's program, for giving me the opportunity to be part of this program and for his assistance in securing this thesis collaboration.

Lastly, I would like to thank all my coworkers and classmates for their friendship, support, and occasional help throughout this journey, which greatly contributed to the successful development of this thesis.

# Resumen

Las organizaciones suelen utilizar sitios web internos tipo wiki para almacenar políticas, documentación de herramientas y otra información relevante. Sin embargo, recuperar información de manera eficiente en estos sistemas puede ser un desafío. Este proyecto tiene como objetivo desarrollar un chatbot impulsado por un modelo de lenguaje natural preentrenado para ayudar a los usuarios a navegar por la wiki. El chatbot procesará consultas de los usuarios, extraerá información relevante y proporcionará respuestas precisas, mejorando la accesibilidad a la información y reduciendo el esfuerzo de búsqueda. Al aprovechar técnicas de procesamiento del lenguaje natural, esta solución busca optimizar la gestión del conocimiento y la experiencia del usuario, garantizando al mismo tiempo la fiabilidad de las respuestas.

**Keywords:** Generación aumentada por recuperación (RAG), Wiki interna, Asistente chatbot, Modelo de lenguaje preentrenado, Consultas de usuarios, Procesamiento de lenguaje natural (NLP).

# Abstract

Organizations often use internal wiki-style websites to store policies, tool documentation, and other relevant information. However, retrieving information efficiently from these systems can be challenging. This project aims to develop a chatbot powered by a pre-trained natural language model to assist users in navigating the wiki. The chatbot will process user queries, extract relevant information, and provide accurate responses, improving information accessibility and reducing search effort. By leveraging natural language processing (NLP) techniques, this solution seeks to enhance knowledge management and user experience while ensuring response reliability.

**Keywords:** Retrieval augmented generation (RAG), Internal wiki, Chatbot assistant, Pre-trained language model, User queries, Natural Language Processing (NLP).

# Table of Contents

# 1 INTRODUCTION

## 1.1 Background of motivations

In today's fast-paced corporate environment, access to accurate and timely information is a cornerstone of efficiency and productivity. Organizations increasingly rely on internal wikis to store and manage essential knowledge, including company policies, procedural guidelines, tool documentation, and onboarding materials. These wikis act as centralized knowledge repositories, enabling employees to locate the information they need to perform their tasks effectively.

Despite their intended purpose, internal wikis are often plagued by usability issues. The sheer scale and complexity of these knowledge bases, combined with limited or unintuitive search functionalities, can make information retrieval a cumbersome and time-consuming process. Employees thus frequently find themselves struggling to locate relevant content, leading to frustration and delays in task execution.

To address these challenges, organizations are turning to artificial intelligence solutions powered by advanced Natural Language Processing (NLP) techniques. One promising approach is Retrieval-Augmented Generation (RAG), which combines two key components: an information retrieval system capable of identifying relevant knowledge from large datasets, and a generative language model capable of synthesizing responses in natural language. By leveraging RAG models, it becomes possible to build intelligent chatbot assistants that integrate directly with an organization's knowledge base, offering precise, contextually relevant, and conversational answers to user queries.

## 1.2 System Overview

### 1.2.1 Problem Statement

While traditional keyword-based search engines provide a rudimentary solution for navigating internal wikis, they often fall short of delivering an efficient and satisfactory user experience, especially in large organizations with extensive and frequently updated knowledge bases. The challenge lies in designing and implementing an intelligent chatbot assistant that can:

-Understand user queries expressed in natural language.

-Efficiently retrieve the most relevant content from the internal wiki.

-Generate accurate, reliable, and contextually appropriate responses.

-Minimize the risk of generating false or misleading information.

-Remember the chat history for a more meaningful discussion, and to decide whether or not it needs to retrieve more information in the wiki to answer the user query

-Enhance the overall user experience by reducing the time and effort required to find information.

To achieve this, the system was conceptually divided into three key components:

### 1.2.2 Information Retrieval

The first major component of the system is a retrieval mechanism capable of accurately identifying and extracting relevant content from a large body of internal documentation. This requires bridging the gap between the way users naturally phrase their questions and the way information is formally written and organized within the wiki. The objective is to return, for each user query, the document fragments most likely to contain or support a correct answer. To ensure the system is viable for internal deployment, the retrieval process should achieve an accuracy of at least 90%, the retrieval accuracy being defined as the proportion of queries for which the correct document is successfully identified and returned.

### 1.2.3 Conversation Management

The second core component centers on how the system interacts with the user. This includes the use of large language models (LLMs) to generate responses based on the retrieved information, as well as techniques to manage multi-turn conversations, maintain context, and control the model's behavior to ensure coherent, relevant, and trustworthy answers. It also encompasses prompt design strategies and the logical flow behind how queries are processed and responded to, forming the backbone of the chatbot's reasoning engine. To ensure the system meets internal usage standards, the chatbot's answers should achieve a user satisfaction rating averaging at least 4 out of 5.

### 1.2.4 Front-End Application

Finally, to make the system accessible and usable, a web-based application will be developed. This front end will allow users to input their queries, view responses, and interact with the assistant in real time. It aims to offer a user-friendly interface that hides the complexity of the underlying system while still providing useful features such as transparency over retrieved sources.

Together, these three components define the high-level architecture of the envisioned solution: an intelligent assistant capable of retrieving and delivering knowledge from an internal wiki through natural conversation.

## 1.3 Current State of Knowledge in RAG techniques

In reality, the core focus of the research in this project will be on information retrieval, that can be considered the most difficult and important part. But before beginning the investigations, let's take a look at the current state-of-the-art retrieval techniques.

Retrieval-Augmented Generation is a cutting-edge approach in artificial intelligence that combines information retrieval with generative language models to create systems capable of producing precise, context-aware responses to queries. Over time, a variety of techniques have been developed to enhance the performance, accuracy, and adaptability of RAG systems. Below is a summary of these techniques, highlighting the current state of knowledge and addressing the challenges involved in selecting and combining the appropriate methods for specific applications.

### 1.3.1 Text Processing

A key element of RAG is the preprocessing of data into manageable units for retrieval, often referred to as *chunking*. Basic approaches divide text arbitrarily, but more advanced methods like semantic chunking create chunks based on meaning, while proposition chunking breaks down content into atomic, factual statements for more targeted retrieval. Context also plays a critical role in ensuring meaningful results. Contextual techniques, such as retrieving neighboring chunks alongside the primary one, provide broader background information for generative models. Compression methods filter out irrelevant data while retaining the most significant parts of the retrieved content, maximizing both efficiency and relevance.

The chunks then go through the embedding phase, which involves converting these pieces of text into high-dimensional numerical vectors using a pre-trained model, such as sentence transformers or other NLP-based embedding models. These vectors (when computed with LLMs) capture the semantic meaning of the text, enabling comparisons between chunks and queries based on their contextual similarity rather than exact keyword matches.

The query itself is another focal point of optimization in RAG systems. Techniques like query transformation rewrite, expand, or decompose user queries to improve retrieval accuracy, while reranking methods reorder retrieved results to prioritize those most relevant to the query. Some systems generate questions from text chunks to supplement retrieval, while others employ dynamic strategies to adapt retrieval based on the type or complexity of the query. These methods aim to ensure that user intent is fully understood and accurately addressed.

### 1.3.2 Dynamic RAG

Dynamic and adaptive capabilities have introduced a new level of sophistication to RAG. Systems like Adaptive RAG and Self-RAG dynamically evaluate query requirements, determine the best retrieval strategies, and assess whether additional context is necessary. These systems can also refine their responses over time by incorporating feedback loops, enabling them to learn and improve continuously. The integration of multiple retrieval methods, such as combining vector-based and keyword-based searches, has further enhanced system robustness. Knowledge graphs allow RAG models to traverse interconnected concepts, adding depth to their retrieval capabilities, while multimodal approaches integrate images alongside text for more comprehensive information retrieval.

In addition to these innovations, reinforcement learning has been explored to optimize RAG systems, allowing them to dynamically maximize performance based on specific objectives, such as improving retrieval relevance or minimizing response time. Hierarchical indices have also been introduced, offering a combination of high-level summaries and detailed content to strike a balance between efficiency and depth.

### 1.3.3 Remaining Challenges

Despite these advancements, significant challenges remain in designing and implementing RAG systems. A central difficulty is selecting techniques suited to the specific problem at hand. Not all methods are equally effective in every context, and determining which techniques can be combined to enhance performance without introducing unnecessary complexity is a delicate balancing act. Moreover, trade-offs between efficiency and simplicity are a persistent concern, as advanced methods often require substantial computational resources and development effort.

This research aims to navigate these challenges, exploring the diverse range of available techniques and identifying those best suited to addressing the complexities of internal corporate wikis. It also seeks to understand how these methods can be combined effectively, balancing the need for robust, accurate results with the practical considerations of system design. The overarching goal is to develop a RAG system that achieves maximum performance while remaining adaptable and efficient.

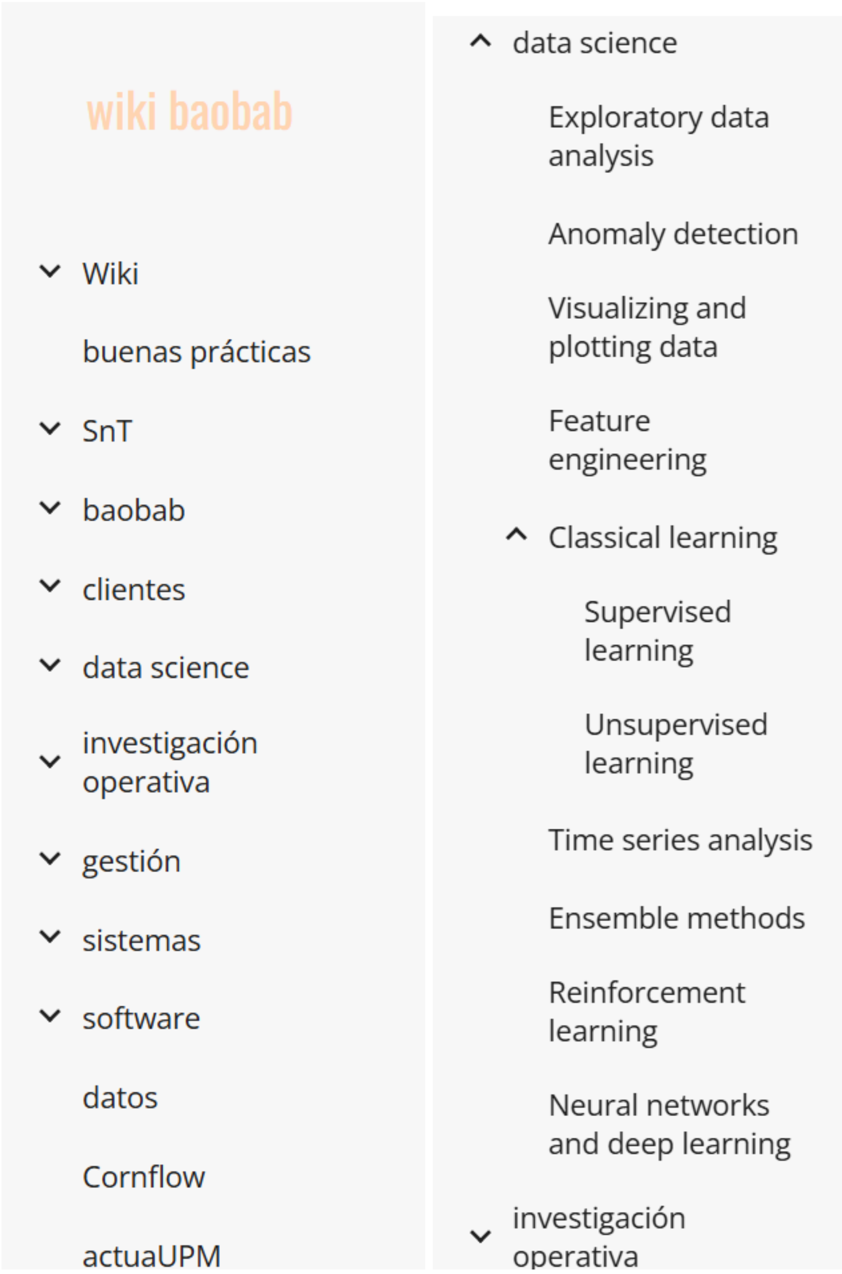
# 2 EMBEDDING-BASED RETRIEVAL

## 2.1 Wiki Structure Extraction

### 2.1.1 An HTML Source

The internal wiki is available as a static repository containing a collection of HTML files, each representing a webpage. The initial step in the processing pipeline involved converting this raw and unstructured HTML data into a format suitable for information retrieval and, ultimately, for input to the language model. The first idea explored was to extract the plain text content of each page and save it as a corresponding text file. This task was relatively straightforward thanks to the `lxml` library, which provides convenient methods for extracting the full textual content from HTML documents.

The real challenge, however, emerged when we sought to enrich the extracted content with structural metadata. In particular, the wiki is not a flat collection of documents, but rather a hierarchical tree originating from a homepage. This homepage links to several sections, which may themselves contain subsections, and so forth. To preserve this hierarchy, we aimed to record, for each page, its full path within the tree. This additional context was expected to enhance the retrieval process by providing relational and organizational cues.




**(a)** *The tree* **(b)** *An expanded section*

**Figure 2.1:** *The wiki tree as visible in every page*

### 2.1.2 Scraping the Tree

Implementing this functionality required reconstructing the wiki's structural hierarchy from the HTML source. Since the tree is embedded and rendered as part of the page layout, it had to be parsed directly from the HTML. Given the recursive nature of the tree and the need to represent nested relationships, we decided to store the resulting structure in a JSON file, as this format naturally supports hierarchical and nested data.

Because the navigation tree appears on every page, it would have been technically possible to reconstruct the hierarchy from any of them. For consistency and simplicity, we chose to extract and parse the tree from the homepage.

To achieve this, a recursive parsing strategy was implemented to traverse the tree of collapsible sections (or *nodes*) that define the wiki's navigation structure. Each section in the wiki is associated with a specific webpage, its "homepage", acting as an overview, and may contain both terminal pages and nested subsections. It is important to clarify that these sections are abstract entities: they exist only as structural elements, and each one has a corresponding "homepage", but are not concrete documents themselves.

The parsing mechanism navigates the HTML document using XPath expressions tailored to the specific patterns observed in the wiki. Each section is processed to extract:

- The name and hyperlink of its associated page (if available).
- The path from the root to the current section's associated page, used as a unique identifier and future storage location.
- A list of direct child pages, each stored with its metadata.
- A recursive call to process subordinate sections, maintaining the hierarchical relationships.


```json
{
    "name": "Root",
    "link": null,
    "path": "Root",
    "children pages": [
        {
            "name": "datos",
            "link": "datos.html",
            "path": "Root/datos.txt"
        },
        {
            "name": "Cornflow",
            "link": "Cornflow.html",
            "path": "Root/Cornflow.txt"
        },
        {
            "name": "actuaUPM",
            "link": "actuaUPM.html",
            "path": "Root/actuaUPM.txt"
        }
    ],
    "children sections": [
        {
            "name": "Wiki",
            "link": "Wiki.html",
            "path": "Root/Wiki",
            "children pages": [
                {
                    "name": "Wiki",
                    "link": "Wiki.html",
                    "path": "Root/Wiki/Wiki.txt"
                }
            ],
            "children sections": [
                {
                    "name": "edición de wikis",
                    "link": "edicin%20de%20wikis.html",
                    "path": "Root/Wiki/edición de wikis",
                    "children pages": [
                        {
                            "name": "edición de wikis",
```


**Figure 2.2:** *The built json-style tree*

**Difficulties**

Several challenges were encountered during implementation. A primary difficulty was the disparity of the HTML structure across pages: the parser had to be fault-tolerant in order to deal with missing or malformed elements. To that end, robust exception handling and logging mechanisms were integrated, ensuring that errors were recorded but did not halt execution.

Another issue involved link filtering. Many nodes in the tree pointed to external pages within the intranet or to unrelated content that could not be analyzed. To avoid polluting the tree with irrelevant entries, only links ending with the `.html` suffix were retained, under the assumption that these represented valid internal wiki pages. The sections overview pages were deliberately included as child pages of their respective sections to ensure that all terminal documents were handled uniformly during later processing steps.

In summary, the tree construction process provides the structural foundation of the entire Retrieval-Augmented Generation system. By transforming the visual interface of the wiki into a machine-readable graph of content nodes, it enables precise document segmentation, contextual metadata preservation, and efficient, structured information retrieval.

## 2.2 Document Embedding

### 2.2.1 Implementation

Following the extraction of the wiki's hierarchical structure and raw textual content, the next step was to implement a retrieval mechanism capable of identifying relevant documents based on user queries. The previously constructed wiki tree allowed us to organize the extracted text files within a directory structure that mirrored the tree's layout. This alignment facilitated consistent access to any given page and preserved its contextual position within the overall knowledge base.

The initial retrieval strategy relied on generating semantic embeddings (high-dimensional vectors that encode the semantic content of a text segment) for each wiki page. These embeddings were generated using a pre-trained large language model (LLM) with embedding capabilities. The core idea was to compute an embedding for the user's query at runtime and compare it with the precomputed document embeddings to estimate semantic similarity. The underlying assumption was that documents with embeddings most similar to the query are more likely to contain relevant information for answering it.

During the embedding phase, each wiki page was read and passed to the LLM, which returned its vector representation. This vector was then saved to disk as a sequence of floating-point values, each written on a separate line. The embeddings were stored in a directory structure identical to that of the text files, ensuring traceability and consistency with the structural metadata defined by the wiki tree.

### 2.2.2 First Limitations

Unfortunately, initial evaluations of the embedding-based retrieval system revealed significant limitations. The system often failed to retrieve pages containing the necessary information to answer specific user queries. This observation led us to hypothesize that the issue stemmed from the granularity of the embeddings—entire wiki pages were being embedded as single semantic units, which proved insufficient for capturing fine-grained information.

Many wiki pages do not focus on a single coherent topic but instead cover multiple, sometimes unrelated, subjects. As a result, embedding the entire page produces a generic vector that tends to obscure more specific content. In cases where a user's question corresponds to a small segment—perhaps just a sentence or paragraph buried within a larger document—the signal of that relevant content is lost in the embedding of the broader, semantically diluted page. Consequently, semantically similar questions and documents often failed to match during retrieval.

To address this limitation, we decided to enhance the scraping phase by enabling the system to handle much smaller and more focused content chunks, at the cost of having much more documents to manage.

## 2.3 Improved Embedding-Based Retrieval

### 2.3.1 Variable Chunk Size

While reducing the chunk size appeared to be a promising strategy to improve retrieval accuracy, it introduced the challenge of determining the optimal chunk granularity. We hypothesized that the most meaningful way to segment the content was to align with the natural human organization of the wiki pages—specifically, by using existing structural elements such as sections, subsections, and paragraphs within pages as the basis for chunk formation.

Given the uncertainty regarding which structural level would yield the best retrieval performance, we opted to construct, for each page, a tree that reflects its semantic hierarchy. In this tree, nodes represent structural elements such as sections and subsections, while the leaves correspond to individual paragraphs. This representation would provide the flexibility to experiment with different levels of granularity during the retrieval process, allowing us to evaluate the impact of chunk size on performance.

However, implementing this approach presented several challenges, the most significant of which was the discrepancy between the structural sections defined in the HTML code and the actual semantic divisions perceived by human readers. In many cases, what constituted a single logical section for a human—such as a block of text accompanied by an illustrative image—was fragmented in the HTML into multiple elements. For example, an image inserted mid-section could result in the HTML being split into two separate section tags, disrupting the intended continuity of the content.

To address this issue, we opted not to define content chunks based solely on HTML section tags, as their structure did not consistently reflect the document's true semantic boundaries. An exception was made for paragraphs, which were reliably represented by HTML paragraph tags and did not exhibit such inconsistencies. Instead, chunk boundaries were defined based on the document's titles and subtitles, which could be clearly identified by their corresponding header tags, and parapraphs.

### 2.3.2 Structured Text Conversion

This strategy involved converting the HTML content into a plain-text format enriched with custom markers for titles and paragraphs. These markers serve as semantic delimiters, designed to be both machine-readable and human-interpretable. To achieve this, the algorithm began by modifying the HTML code itself, directly annotating the text content of key structural elements—namely titles and subtitles—with custom tags. These tags were inserted around the corresponding header elements to ensure their visibility and hierarchical interpretation in the plain-text output.

At the top of the hierarchy, the page title was emphasized and inserted at the beginning of the document using a standardized marker. Likewise, all titles and subtitles, identified based on their header level (e.g., `<h1>`, `<h2>`, `<h3>`), were transformed into uppercase and surrounded by boundary tags. The visual format of these tags varied depending on the heading level, thereby preserving the hierarchical relationships between sections.

However, this direct tagging strategy could not be applied to paragraphs, even though they were easily identifiable in the HTML due to their specific tag (<p>). Unlike titles, the text content of paragraphs was often nested within additional tags, making it infeasible to insert custom markers directly into the HTML without risk of disrupting the internal structure. This limitation necessitated a more sophisticated, two-stage approach.

In the first stage, the entire text content of the page—now containing the annotated titles but unmarked paragraphs—was extracted from the HTML and stored in a text file. The second stage involved iterating through each paragraph element in the original HTML, extracting its textual content, and attempting to locate it within the previously generated plain-text document. When a match was found, and the paragraph exceeded a predefined length threshold (to exclude trivial content), paragraph-specific tags were inserted around the matched text. This ensured that paragraph-level segmentation could still be achieved despite the structural limitations of the HTML.


**Figure 2.3:** *The wiki home page*

```
<T>--------WIKI--------<T/>

<T>--RECOMENDACIÓN GENERAL--<T/>
Los idiomas de la wiki son el español o el inglés.<P>Conviene escribir en aquello que requiera menor esfuerzo y que facilite la lectura pos
<P>No se recomienda traducir al español si incorporamos alguna referencia (convenientemente referenciada) cuyo texto original es en inglés.
<P>Con normal general, para la elaboración de textos completamente originales, se recomienda redactar en el idioma en el que quien escriba

<T>------PRESENTACIÓN------<T/>
<P>La wiki, Google Drive y bitbucket son las fuentes corporativas de información en baobab y, en particular, de toda la información de cará
<P>La wiki es el lugar donde debe quedar descrito el conocimiento de baobab y donde deben quedar recogidos los aspectos más importantes de
<T>--BITBUCKET, PROYECTO DE FORMACIÓN--<T/>
<P>Este proyecto contiene repositorios de código fuente de aplicaciones de diferente naturaleza y con diferente tamaño. Puede haber desde s
<T>--LA UNIDAD DE RED DE INTERNORECURSOS--<T/>
<P>Esta unidad de red tiene una estructura de carpetas muy parecida al menú de navegación de la wiki y contiene presentaciones, documentos
<T>--FUENTES EXTERNAS --<T/>
<P>Como repositorios de terceros, cursos online, foros, etc.<P/>

<T>----FLUJO DE TRABAJO----<T/>
<P>Cuando alguien quiera saber algo, sería deseable que acuda a la wiki. Si la información que hay disponible es insuficiente, podría hacer
Además, para fomentar el uso de la wiki:<P>Cuando alguien modifique la wiki registrará los cambios realizados en el wiki-log. En cada entra
<P>Con esto, será posible realizar notificaciones periódicas con los últimos cambios.<P/>
<P>Naturalmente, cualquier podrá entrar en el wiki-log y revisar las últimas actualizaciones o conocer quién modificó alguna sección en par
<P>Una de las funcionalidades más valiosas de la wiki es la búsqueda. A pesar de que la wiki debe mantenerse ordenada, la herramienta de bú
<P>Para aprovechar aún más la funcionalidad anterior, debemos mantener enlazadas las entradas de la wiki a los repositorios correspondiente
```

**Figure 2.4:** *The structured text generated from the home page*

### 2.3.3 Hierarchical Conversion to JSON Format

Following the annotation of each wiki page with semantic markers for titles and paragraphs, the structured plain-text content must be transformed into a format that explicitly encodes the document hierarchy. To achieve this, the annotated text is recursively parsed and converted into a nested `JSON` structure that reflects the internal organization of the page.

The transformation process relies on the hierarchical cues introduced during the text annotation phase. Specifically, the algorithm interprets the custom title markers—whose formatting indicates their depth in the document structure—as delimiters that define parent-child relationships between sections. At the top level, the page title is extracted and used to initialize the root of the hierarchy. Each subsequent subsection is identified based on its associated heading tag (e.g., visualized through a decreasing number of dashes in the marker), and is recursively parsed in a similar manner.

For each structural unit detected (i.e., section or subsection), a unique title and path are generated. The path is used as a persistent identifier and reflects the position of the unit within the document tree. The path is constructed incrementally by concatenating the normalized titles of parent sections, using lowercase formatting and replacing whitespace with underscores. This convention ensures both readability and compatibility with file systems or indexing structures.

When the algorithm reaches the deepest levels of the hierarchy—typically the leaf nodes corresponding to paragraphs—it extracts each paragraph marked with the dedicated tags and stores them as individual entries. Each paragraph is assigned a unique name (e.g., `paragraph_0`, `paragraph_1`, etc.), along with its relative path and text content. This level of granularity enables precise retrieval of fine-grained content units when needed.

To enable flexible chunk size selection during retrieval, each node in the hierarchy maintains a list of numerical values referred to as "parenting levels." These values capture the depth of the node within the document structure by representing the set of distances to all leaf nodes that descend from it. In other words, they indicate how far the current node is from its most granular content units, thereby allowing the system to make informed decisions about the level of detail to include during retrieval.

The output of this process is a nested dictionary structure that is serialized into `JSON`. Each node contains the following elements:

- `title`: the name of the section or paragraph.
- `path`: a unique identifier derived from its hierarchical position.
- `parenting levels`: a list indicating the content's depth in the structure.
- `text content`: the raw, unannotated text associated with the node.
- `summary`: a placeholder for future inclusion of model-generated summaries.
- `children list`: a list of recursively structured sub-sections or paragraphs.


```
{
    "title": "WIKI",
    "path": "wiki",
    "parenting levels": [
        3,
        4
    ],
    "summary": "summary",
    "text content": "WIKI\n\nRECOMENDACIÓN GENERAL\nLos idiomas de la wiki son el español o el inglés.Conviene escribir en aquello que requ
    "children list": [
        {
            "title": "Intro",
            "path": "wiki/intro",
            "parenting levels": [
                2
            ],
            "summary": "summary",
            "text content": "Intro\nRECOMENDACIÓN GENERAL\nLos idiomas de la wiki son el español o el inglés.Conviene escribir en aquello q
            "children list": [
                {
                    "title": "RECOMENDACIÓN GENERAL",
                    "path": "wiki/intro/recomendación_general",
                    "parenting levels": [
                        1
                    ],
                    "summary": "summary",
                    "text content": "RECOMENDACIÓN GENERAL\nLos idiomas de la wiki son el español o el inglés.Conviene escribir en aquello
                    "children list": [
                        {
```


**Figure 2.5:** *The json file generated from the home page*

This hierarchical representation of content enables the system to navigate and retrieve information not only at the page level, but also at the section, subsection, or paragraph level.

### 2.3.4 Embedding Generation and Storage

Once the wiki content had been transformed into a structured JSON format that captured the hierarchical organization of sections, subsections, and paragraphs, the next phase involved converting these text segments into their corresponding vector embeddings.

The embedding generation process was designed to operate selectively across different levels of the document hierarchy. This flexibility was made possible through the `parenting levels` attribute, previously computed during the hierarchical structuring phase. By specifying a target parenting level—corresponding, for instance, to sections, subsections, or individual paragraphs—the system could generate embeddings with fine-grained control over the semantic resolution at which retrieval was performed.

Embeddings were once again stored on disk following a structured naming and directory convention. Each embedding was saved as a plain-text file (same format as before), within a directory that corresponded to its parenting level (e.g., `level_0`, `level_1`, etc.). Any node whose `parenting levels` list included the specified level would have its associated text content converted into an embedding and stored within the appropriate directory. The internal structure of these directories mirrored the hierarchical paths used in the JSON representation.

To further support efficient indexing and retrieval, the system also generated a set of address files. For each parenting level, a text file was created listing the hierarchical paths of all nodes with embeddings at that level. These address files served as lookup tables, enabling the system to quickly identify and locate the relevant embeddings during query processing.

When processing a user query, the system requires the specification of one or more target parenting levels for retrieval—either explicitly provided or set to a default configuration. This enables the retrieval process to operate at multiple levels of granularity simultaneously, accommodating both broad and fine-grained information needs. The system begins by reading the address files corresponding to the selected levels, then loads the associated embeddings from disk. These embeddings are subsequently compared to the embedding of the query to identify the most semantically similar content (using cosine similarity between vector embeddings) across the designated hierarchical depths.

This embedding phase effectively transformed the semantically enriched content tree into a searchable vector space, indexed by hierarchical depth. It enables multi-resolution semantic search, allowing the system to retrieve information at the level of abstraction best suited to the user's intent.

## 2.4 Limitations

Despite the enhancements made to the retrieval pipeline—such as semantic chunking and hierarchical embedding generation—the system's overall performance still fell short of expectations. Upon closer examination, it became evident that the remaining issues were not primarily due to flaws in the system's architecture or the choice of chunk granularity. Instead, they stemmed from more fundamental limitations inherent to the use of vector embeddings themselves, making them significantly more challenging to resolve.

### 2.4.1 A Language Issue

A key issue identified was the inconsistency in embedding quality between languages. The model used for embedding generation (DeepSeek, chosen for being among the most capable freely available options) demonstrated significantly better performance in English than in Spanish—the primary language of the wiki content and the intended target language of the system. For instance, the semantic similarity score between the English words *eat* and *drink* reached 0.74, whereas their Spanish equivalents, *comer* and *beber*, only reached 0.66. Given that similarity scores range from -1 to 1 and retrieval decisions often depend on minute decimal differences, this discrepancy posed a serious barrier to effective retrieval in Spanish.

To mitigate this issue, we considered translating the entire knowledge base into English and operating the system in English instead. This strategy involved storing a parallel structure of documents—mirroring the existing Spanish hierarchy—while dynamically translating incoming user queries from Spanish to English. Retrieval would then proceed using the same embedding and comparison mechanisms already in place.

However, this approach introduced a complex trade-off. One possibility was to translate the content at the same level of granularity used during embedding—i.e., section by section or paragraph by paragraph. While this aligned well with the system's structure, it risked losing critical context, particularly for lower-level chunks such as paragraphs, which often consist of only one or two sentences and cannot be meaningfully interpreted in isolation.

An alternative was to translate entire pages using the structured plain-text files previously generated from the HTML source. These full-page translations would preserve the original structure and context, making them ideal for re-generating the hierarchical JSON files. We attempted this method by prompting a large language model (DeepSeek again) to translate the full contents of each structured file while preserving our custom formatting and semantic markers. Although this strategy was successful on many pages, it encountered critical failures on others—particularly those that were excessively long or contained embedded programming code. In such cases, the model often misinterpreted the custom markers or confused them with code syntax, resulting in corrupted or lost structural formatting.

Since the hierarchical JSON files depended entirely on the integrity of the structural markers in the translated files, these failures rendered the affected pages unusable for indexing and retrieval. As a result, this solution, while conceptually sound, could not be reliably applied across the full dataset.

These findings highlighted a significant limitation of embedding-based retrieval systems operating in non-English contexts, especially when relying on models primarily trained and evaluated in English.

### 2.4.2 Semantic Misalignment

A further limitation encountered during the development of the retrieval system stemmed from the way large language models (LLMs) interpret semantic similarity. In many cases, the models' understanding of alignment between texts differs substantially from human intuition. One striking observation is that LLM-generated embeddings appear to prioritize surface-level features—such as sentence structure—over actual semantic content. For example, the question *"How to create a contact in the CRM?"* yields a higher similarity score when compared to another question with a similar syntactic form, such as *"How to create a Toggl register?"* (score: 0.82), than when compared to a sentence that directly answers the original query, like *"To create a contact in the CRM, go to the intranet and [...]"* (score: 0.71). While the latter is clearly a more semantically relevant match from a human perspective, the embedding model ranks the former higher due to structural similarities.

This discrepancy between syntactic and semantic similarity is a well-documented limitation in Retrieval-Augmented Generation systems. A commonly proposed solution is to reformulate document embeddings by generating representative questions that the document could answer. These synthetic questions are then embedded in place of the raw document content. While this technique can increase alignment with natural queries, it introduces a new set of challenges.

Firstly, generating a meaningful question for every document is non-trivial. Some documents—especially those that are descriptive, instructional, or technical—may not lend themselves easily to a clear, question-like formulation. More importantly, many documents, particularly longer ones such as full pages or sections, are capable of answering multiple diverse queries. This forces the system designer to select an arbitrary number of synthetic questions per document. A fixed number per document risks omitting important queries for longer texts or including irrelevant ones for shorter ones. Alternatively, imposing only a maximum number of questions leads to variability across the dataset, complicating the retrieval logic.

In either case, a common method to compute document-query similarity is to compare the query embedding to all the synthetic questions associated with a document and take the average similarity score. However, this approach can unfairly penalize large documents: if a document contains many questions but the query aligns with only one of them, the resulting average may be too low to ensure retrieval, despite semantic relevance.

An inverse strategy has also been proposed: instead of modifying the documents, generate hypothetical documents in response to each query, and then compare these to the original database entries. However, this method is also problematic. Since the generated documents are created without access to the real database, they are often speculative, hallucinated, or stylistically misaligned with the actual documents, thereby weakening the effectiveness of the embedding comparison.

### 2.4.3 Embeddings Instability

Ultimately, what led us to seriously question the viability of relying on embedding-based retrieval—at least with current open-source models—was the discovery of embedding instability. For instance, the semantic similarity between *"How to create a contact in the CRM?"* and *"How to create a Toggl register?"* drops from 0.82 to 0.62 simply by adding a trailing space to the second sentence. A 0.20 change in alignment due to such a trivial modification highlights the fragility and unpredictability of the embedding process. When similarity scores are often distinguished by hundredths of a point, such volatility undermines the trustworthiness of the entire retrieval system.

These observations made it increasingly clear that vector-based retrieval—particularly when using free or general-purpose models—was insufficiently reliable for the demands of this system. Consequently, we were led to explore a fundamentally different approach to information retrieval, which will be discussed in the following chapter.

3

# TF-IDF Retrieval

## 3.1 A Simpler Alternative

In light of the limitations observed with embedding-based retrieval, we decided to explore a more traditional, interpretable, and deterministic approach to information retrieval: the TF-IDF technique.

TF-IDF, which stands for *Term Frequency - Inverse Document Frequency*, is a widely used method in information retrieval and text mining. It quantifies the importance of a word within a document relative to a larger corpus. Specifically, the term frequency (TF) measures how often a word appears in a given document, while the inverse document frequency (IDF) downscales the importance of terms that are common across many documents. The TF-IDF score is computed as:

$$\text{TF-IDF}(t, d, D) = \text{TF}(t, d) \times \text{IDF}(t, D) = \left(\frac{f_{t,d}}{\sum_{t'} f_{t',d}}\right) \times \log\left(\frac{N}{|\{d \in D : t \in d\}|}\right)$$

where $f_{t,d}$ is the raw frequency of term $t$ in document $d$, $N$ is the total number of documents in the corpus $D$, and the denominator of the IDF is the number of documents in which the term $t$ appears.

The product of TF and IDF highlights terms that are both representative of the document and distinctive within the overall collection. In this way, TF-IDF provides a numerical representation of textual content that emphasizes informative and discriminative vocabulary.

Unlike dense vector embeddings produced by large language models, which operate in high-dimensional semantic spaces and often lack interpretability, TF-IDF vectors are sparse, easy to compute, and directly linked to the actual words in the document. This transparency not only improves the traceability of results but also reduces the risk of unexpected behavior, such as misalignments due to syntax or unstable similarity scores caused by minor textual variations.

Another key advantage of TF-IDF is its language neutrality: since it relies purely on term statistics, it does not depend on the quality of a language model's training in a particular language. This makes it particularly suitable for our use case, where the majority of the content is written in Spanish and where pre-trained embeddings have demonstrated significant performance degradation compared to their English counterparts.

While TF-IDF may not capture deep semantic relationships as effectively as neural embeddings, it offers a level of robustness, simplicity, and interpretability that makes it an attractive alternative for document retrieval. In this chapter, we describe how TF-IDF was integrated into our retrieval pipeline, how documents were indexed and queried, and how this approach compares in practice to the previous embedding-based method.

## 3.2 Implementation and Processing Pipeline

To operationalize the TF-IDF approach for document retrieval, we designed and implemented a modular processing pipeline capable of vectorizing content at multiple levels of granularity.

### 3.2.1 Corpus Construction

The process begins with the preparation of the corpus from which TF-IDF vectors are generated. For each document—be it a full page or a hierarchical unit such as a section or paragraph—the raw text is extracted, normalized and stored in hierarchical JSON files exactly the same way it was done in the second version of the embedding-based retrieval pipeline. To compute the vector representation of the text chunks, the system then traverses the JSON file of each page and extracts the content associated with a specified parenting level.

Rather than relying on the full verbatim text of each document, the system focuses on enhancing semantic relevance by extracting and lemmatizing nouns. This decision was motivated by the assumption that nouns are typically the most informative elements for determining the subject of a document, and thus, the most useful for identifying relevant matches during retrieval. By restricting the vocabulary to lemmatized nouns, the system significantly reduces the dimensionality of the TF-IDF matrix, avoiding the computational and memory overhead associated with a very large and sparse feature space.

However, implementing this approach required addressing the multilingual nature of the wiki, which contains content in both Spanish and English. No single library was capable of performing accurate noun recognition and lemmatization across both languages simultaneously. To resolve this, we designed a two-stage processing pipeline.

In the first stage, Spanish nouns are identified and lemmatized using Spacy's `es_core_news_lg` model, which provides robust support for Spanish part-of-speech tagging and lemmatization. Any remaining words not recognized as valid Spanish nouns were passed through a second validation step using WordNet, a lexical database for English. Words confirmed by WordNet as English nouns were retained and lemmatized accordingly. This dual-pass strategy ensured that informative terms from both languages were captured, while minimizing the inclusion of irrelevant or ambiguous tokens.

To further improve topic representation—particularly in short documents such as paragraphs—the titles associated with each node's hierarchical path were also appended to the document text multiple times (typically two or three repetitions). This technique reinforced the importance of title words, which are often highly indicative of the document's subject matter and context within the overall wiki structure.

### 3.2.2 Vectorization

The core of the system relied on the `TfidfVectorizer` from `scikit-learn`. Documents were tokenized into their lemmatized noun forms and transformed into sparse TF-IDF vectors. For each processed corpus (one corpus per parenting level, e.g. page, section, subsection, paragrpah...), the resulting TF-IDF matrix was stored in a directory structure corresponding to the parenting level of the content. For example, embeddings for full pages were saved under `level_page`, while embeddings for sections or subsections were stored under `level_1`, `level_2`, etc.

Each term in the vocabulary was stored separately: for every word, a plain-text file was created where each line contained its TF-IDF score for a document. This format allowed for efficient access to term-document weights, since it only needed to load the files corresponding to the words contained in a query to process it, instead of having to load the full matrix into memory.

To facilitate fast lookup and maintain alignment between terms, documents, and vector indices, a metadata dictionary, stored in a json file, was also generated for each corpus. This dictionary included:

- `index to word` and `word to index`: mappings between vocabulary terms and their index in the TF-IDF matrix (their corresponding column).
- `index to corpus element` and `corpus element to index`: mappings between document paths and their index in the TF-IDF matrix (their corresponding row).

The system was designed with flexibility in mind. TF-IDF embeddings could be generated at any desired depth of the content hierarchy by specifying the target `parenting level`. Overall, this TF-IDF implementation provided a lightweight, language-agnostic, and fully interpretable alternative to neural embeddings.

## 3.3 Improvements

Although the TF-IDF-based retrieval system proved more robust and interpretable than its embedding-based predecessor, it was not without significant limitations. One fundamental constraint of the TF-IDF method lies in its reliance on exact term matching: the similarity between a query and a document depends entirely on the presence of shared words. This becomes problematic when users formulate queries using vocabulary or grammatical forms that differ from those used in the documents, even when both expressions refer to the same concept.

### 3.3.1 Grammatical variations

For example, a user might ask, *"¿Cómo crear un contacto en el CRM?"* (How to create a contact in the CRM?), while the relevant document contains the phrase, *"La creación de contactos en el CRM se hace mediante..."* (The creation of contacts in the CRM is done through ...). Despite the obvious semantic correspondence, standard TF-IDF may fail to match these texts effectively, as it does not recognize that *crear* (to create) and *creación* (creation) refer to the same underlying concept.

To address this mismatch, we expanded our preprocessing pipeline to incorporate both grammatical flexibility and semantic enrichment. First, we moved beyond restricting our vocabulary to nouns and began including verbs as well.

Additionally, we introduced a novel step that leveraged a large language model to perform bidirectional grammatical transformations. Specifically, the system identified nouns and verbs within each document (the same way it used to identify nouns) and used an LLM to generate their corresponding grammatical counterparts—transforming nouns into related verbs and vice versa, when applicable. This approach ensured that both *crear* and *creación*, for instance, were included in the processed vocabulary, greatly improving the likelihood of matching related concepts expressed in different grammatical forms.

### 3.3.2 Synonym research

Beyond grammatical variants, the system also accounted for lexical diversity through synonym expansion. This was achieved by querying the WordReference online synonym dictionary via a custom scraping function. For each word in a document, a list of semantically related alternatives was retrieved to enrich the vocabulary.

However, as previously discussed in the context of question generation for documents, this approach introduces variability: different words yield different numbers of synonyms depending on their lexical richness and the entries available in WordReference. This variability can lead to inconsistency in vector representations, potentially biasing the retrieval process toward documents associated with more synonyms.

To mitigate this issue, a fixed number of synonyms was retained for each word. When fewer synonyms were available, the original word was repeated to maintain consistency in input length. This ensured uniformity across documents while avoiding excessive expansion that could introduce semantic noise.

Rather than selecting synonyms at random, which could weaken document coherence, the system employed a large language model to rank and filter the list of retrieved synonyms. The model was provided with the original document context and the full list of candidate synonyms, and was tasked with selecting the most appropriate alternatives. By leveraging the LLM's contextual understanding, only the top-ranked synonyms were preserved, thereby ensuring that synonym expansion enhanced recall without diluting the semantic focus of the document.

These semantic and lexical expansions were applied consistently to both documents and user queries before TF-IDF vectorization. As a result, the system became less sensitive to variations in phrasing, word choice, and grammatical structure—bridging the gap between how users formulate questions and how knowledge is represented in the wiki content.

## 3.4 TF-IDF limitations

Despite the series of enhancements applied to the TF-IDF-based retrieval system, it became increasingly evident that some limitations are inherent to the TF-IDF approach itself and are difficult, if not impossible, to fully overcome through preprocessing alone.

At its core, TF-IDF operates purely on word frequency statistics and is entirely agnostic to semantic context. It lacks any understanding of meaning beyond surface-level term occurrences. This presents several challenges, especially in a linguistically rich and morphologically complex language like Spanish.

One major issue arises from lexical ambiguity and morphological overlap. For example, the Spanish word *creo* can derive from two entirely different verbs: *crear* (to create) and *creer* (to believe), depending on the context. TF-IDF, being unaware of grammatical function or sentence intent, cannot distinguish between these interpretations. Similarly, high-frequency verbs such as *pasar* (to pass, to happen) or *quedar* (to remain, to agree, to be located) have numerous meanings that vary significantly depending on how they are used in a sentence. TF-IDF treats all instances of these terms as equivalent, leading to retrieval matches that are technically valid by word overlap but semantically irrelevant.

Another critical limitation is that TF-IDF does not account for word order. It treats documents as unordered collections of terms (a "bag of words"), which ignores the syntactic structure that is often essential to understanding meaning. For example, the user query *"cómo registrar el tiempo en Toggl?"* (how to log time in Toggl?) could be matched to a document containing the sentence *"crear registros en tiempo real en Toggl"* (create real-time records in Toggl). While both expressions share many of the same words, their meanings diverge significantly. The former refers to time tracking, while the latter is about real-time data creation—two fundamentally different concepts. Yet, under TF-IDF, the high lexical overlap could cause such a document to be erroneously ranked as relevant.

These examples illustrate the fundamental weakness of TF-IDF: it can match words, but not meaning. While our various preprocessing techniques helped reduce the impact of this limitation, they could not eliminate it entirely. As a result, we concluded that TF-IDF—despite being a relatively effective and computationally efficient baseline—could not serve as the sole mechanism for document retrieval in our system.

In the following chapter, we build upon the insights and tools developed thus far to design a new retrieval architecture.

# 4 The Final Architecture

## 4.1 An Hybrid Retrieval

In light of the limitations identified in the TF-IDF-based retrieval system, we opted to design a hybrid architecture that combines both TF-IDF and embedding-based retrieval methods. These two approaches are highly complementary: TF-IDF excels in precision for syntactically well-formed and term-specific queries, while embedding-based retrieval offers superior semantic generalization and robustness to paraphrasing. By integrating both, the system is able to compensate for the weaknesses of each method and capitalize on their respective strengths.

### 4.1.1 The Retrieval Pipeline

This section describes the end-to-end retrieval pipeline of the final system. The process begins with the user query, which is first translated into Spanish using a dedicated LLM. This translation step is applied regardless of the original language of the query, as the current implementation only supports retrieval in Spanish.

Following translation, the query is passed to a second LLM whose role is to perform query expansion. This model generates a set of five reformulated versions of the original query, each designed to preserve the intent while introducing lexical and structural variation. These reformulations leverage synonyms, paraphrasing, and changes in specificity or granularity. The LLM also handles acronym expansion and inclusion, ensuring that both abbreviated and full-form expressions are covered. The purpose of this step is to increase the expressiveness of the query and reduce the risk of mismatch with relevant documents that may be phrased differently. This is particularly beneficial for TF-IDF retrieval, which is sensitive to exact term matching, but also enhances embedding-based retrieval by providing more context-rich and linguistically diverse input.

Moreover, this step addresses cases where the user's original query may be poorly expressed, ambiguous, or overly narrow. Through reformulation, the system can uncover related information needs that the user may not have explicitly articulated, thereby broadening the scope of potentially relevant results and increasing the overall informativeness of the retrieval process.

Each of the reformulated queries is then vectorized using both the TF-IDF method and a pre-trained embedding model. These representations are compared against the precomputed vectors of the documents in the database—also generated using both methods—yielding two independent similarity rankings for each query. For every reformulated query, the system retrieves the top three most similar documents according to TF-IDF and the top three according to embedding-based similarity, resulting in a total of six candidate documents per query.

This process yields up to 30 candidate documents (5 queries × 6 results). To consolidate the results, the system ranks the retrieved documents by frequency of appearance across all reformulations and similarity methods. The three most frequently retrieved documents are then selected as the final output. This voting-based aggregation helps reinforce consensus across both retrieval modalities and across linguistic variations of the query, ensuring the most relevant and robust results are surfaced.

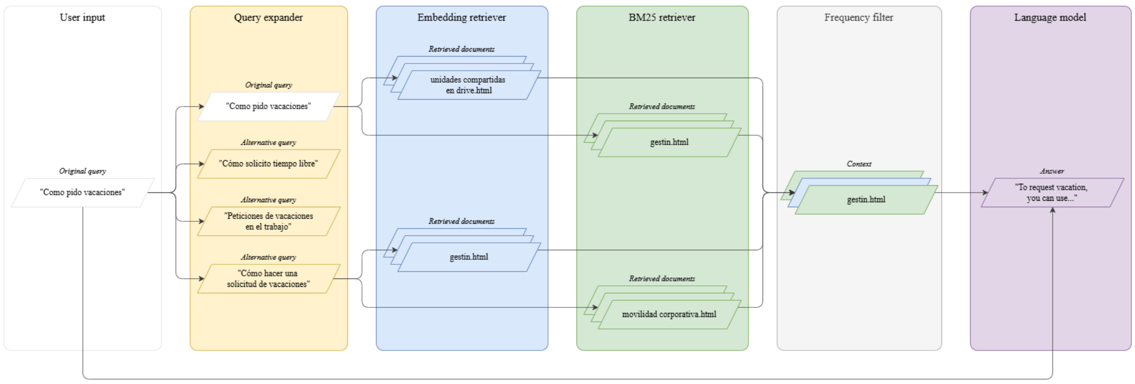



**Figure 4.1:** *The retrieval pipeline*

### 4.1.2 Technical Improvements

The redesign of the system's architecture also provided an opportunity to implement several important technical enhancements and simplifications. One of the most impactful changes was the replacement of earlier, free embedding models with embeddings generated by Gemini, Google's proprietary large language model. As discussed in a previous chapter, free models such as DeepSeek exhibited inconsistencies and instability in semantic similarity scoring—issues that significantly hindered the performance and reliability of the embedding-based retrieval. In contrast, Gemini embeddings delivered markedly more stable and accurate results at a relatively low cost, justifying their integration into the final pipeline.

As a result, all tasks that rely on LLMs are now handled uniformly by Gemini. These tasks include query translation, query expansion, grammatical transformation (e.g., verb-noun conversion), semantic synonym filtering, and embedding generation, as well as other tasks that will be discussed in the following section. Centralizing these operations under a single, high-quality model improves consistency across the pipeline and simplifies model management.

In parallel, we also upgraded the TF-IDF retrieval component by replacing the classical TF-IDF formula with BM25—a state-of-the-art ranking function commonly used in modern search engines. BM25, or Best Matching 25, builds on TF-IDF by introducing term saturation and document length normalization. This means that repeating the same keyword in a document has diminishing returns (addressing keyword spamming), and longer documents are not unfairly penalized or rewarded. Formally, BM25 assigns a score to a document $D$ given a query $Q$ as follows:

$$\text{score}(D, Q) = \sum_{t \in Q} IDF(t) \cdot \frac{f(t, D) \cdot (k_1 + 1)}{f(t, D) + k_1 \cdot (1 - b + b \cdot \frac{|D|}{\text{avgdl}})}$$

Where: - $f(t, D)$ is the frequency of term $t$ in document $D$, - $|D|$ is the length of document $D$, - avgdl is the average document length in the corpus, - $k_1$ and $b$ are hyperparameters that control term frequency saturation and length normalization, respectively.

This improved scoring method was integrated without altering the preprocessing pipeline described in the previous chapter. Lemmatization, synonym expansion, grammatical enrichment, and contextual reinforcement through title repetition continue to be applied prior to vectorization.

Furthermore, to enhance maintainability, modularity, and compatibility with state-of-the-art LLM tooling, the entire system was migrated into the LangChain ecosystem. LangChain is a Python framework specifically designed for building applications that integrate large language models, with robust support for tasks such as retrieval-augmented generation, conversational agents, memory management, and advanced document indexing.

While the original scraping logic remains unchanged, we opted to simplify the downstream processing by treating entire wiki pages as individual chunks, rather than segmenting them into sections or paragraphs. This decision was made feasible by the superior semantic resolution of the Gemini embeddings, which can now handle longer and more diverse text segments without significant degradation in retrieval quality.

After scraping, all wiki pages are converted into LangChain `Document` objects, which are then processed by two vectorization engines: the `BM25Retriever` for BM25-vectorization and `GoogleGenerativeAIEmbeddings` along with `FAISS` for embedding generation and storage. FAISS (Facebook AI Similarity Search) is a highly optimized open-source library for efficient similarity search and clustering of dense vectors. It allows for fast nearest-neighbor search even across large corpora, making it well-suited for real-time semantic retrieval tasks.

Unlike the previous manual storage of individual vectors, the new system stores all document vectors in large, compressed matrices—one for BM25 and one for embeddings—using a single file per method. This approach improves efficiency, reduces disk usage, and increases safety and compatibility by leveraging mature, well-supported open-source tools.

Finally, document retrieval is now handled entirely through LangChain's native retrieval methods. Queries are passed through the appropriate retriever class (BM25 or FAISS), which automatically returns the top-k relevant documents based on the corresponding similarity metric. This integration streamlines the retrieval logic and ensures that the system adheres to current best practices in LLM-assisted search applications.

## 4.2 The Chatbot

### 4.2.1 A Graph-Based Chatbot

After developing an efficient and reliable retrieval mechanism, we turned our attention to the next major challenge—and indeed, the ultimate objective of this project: the design and implementation of a functional and intelligent chatbot assistant.

Basic chatbot architectures are typically linear or monolithic in nature. In such systems, a user query is processed through a fixed sequence of steps—such as preprocessing, retrieval, and response generation—regardless of the context of the conversation or the nature of the question. This rigid pipeline lacks adaptability and often results in suboptimal behavior. For instance, trivial or conversational queries may trigger unnecessary document retrievals, while complex questions requiring external information might be inadequately addressed due to insufficient context handling.

To overcome the limitations of linear chatbot architectures, we adopted a graph-based design using the LangGraph framework from LangChain. This approach offers several critical advantages that align with the demands of a retrieval-augmented, LLM-driven assistant:

- **Flexibility**: The system dynamically adapts to the query, avoiding unnecessary retrievals and providing quicker responses for simple inputs.
- **Reliability**: The system is capable of evaluating whether the retrieved context is sufficient to answer the user's query, improving the overall robustness of interactions.
- **Modularity**: Each functional component is implemented as an independent node. This separation of concerns makes the system easier to extend, maintain, and debug.
- **Control**: Fine-grained control over decision logic and data flow enables performance tuning and predictable behavior across a variety of user interactions.
- **Integration**: The architecture supports the use of multiple LLMs and retrieval strategies, allowing the assistant to handle a wide range of user queries with improved adaptability.

This architectural decision was guided by the need for a system that is not only extensible and interpretable, but also capable of dynamically selecting between two core reasoning modes: direct response generation and context-dependent retrieval. In this design, the chatbot operates as a stateful computation graph. Each node in the graph performs a specific function—such as retrieving documents, evaluating query complexity, translating responses, or generating output—while the edges represent the logic that governs transitions between these operations. The current state of the conversation (including messages, retrieved context, and auxiliary metadata) is propagated through the graph, allowing the system to determine in real time how best to process a user input.

By leveraging this graph-based approach, the assistant becomes capable of adapting to varying levels of query complexity and conversational flow. This results in interactions that are more responsive, contextually appropriate, and aligned with user expectations.

### 4.2.2 The Graph Implementation

The core chatbot pipeline is modeled as a LangGraph, with nodes representing functional components and edges representing decision logic. The graph processes and updates a shared internal state, defined as a Python dictionary with three key fields:

- `messages`: a list of conversation messages (user and assistant turns).
- `context`: retrieved documents from the knowledge base.
- `additional_info`: metadata including query reformulations and retrieval diagnostics.

LangGraph also provides composable mechanisms to manage state over time. For instance, the system includes a custom `forget_gate` function that prunes the document context to the most recently used documents, reducing memory overhead while maintaining conversational relevance. Context accumulation across turns is managed through a custom reducer function (`add_context`), which concatenates document lists as needed.

### 4.2.3 The Conversation Flow

The flow of the conversation begins at the `START` node. From here, a conditional branching function evaluates whether the user's current message requires external knowledge from the wiki or can be answered based solely on recent context. This determination is made by an LLM, using a specialized evaluation prompt that examines the last assistant response, the current user query, and the current retrieved context. If the LLM responds with "No", the system proceeds directly to the chatbot node for response generation; otherwise, it activates the retrieval node.

In the `retrieval` node, the system performs multi-query document retrieval using both TF-IDF (BM25) and embedding-based methods, as previously described. Query expansion and reformulation are conducted using a dedicated LLM before vectorization. Retrieved documents are then added to the conversation state, along with metadata such as which subqueries retrieved which documents.

The `chatbot` node is responsible for generating the assistant's final response by combining the most recent user query with the retrieved contextual information. This process is guided by a carefully designed prompt that shapes both the tone and behavior of the assistant. For simple queries—such as greetings or meta-questions like "What can you do?", the assistant responds directly in a conversational and informal tone. For information-seeking queries, it draws exclusively from the retrieved documents to construct a fact-based, contextually grounded answer. Crucially, when the retrieved context is insufficient to provide a reliable response, the assistant explicitly communicates this limitation to the user. In such cases, it may instead offer a light anecdote, historical reference, or harmless joke to maintain engagement. This approach ensures that the assistant does not invent or speculate on missing information, thereby significantly reducing the risk of hallucinations and enhancing the trustworthiness of its responses.

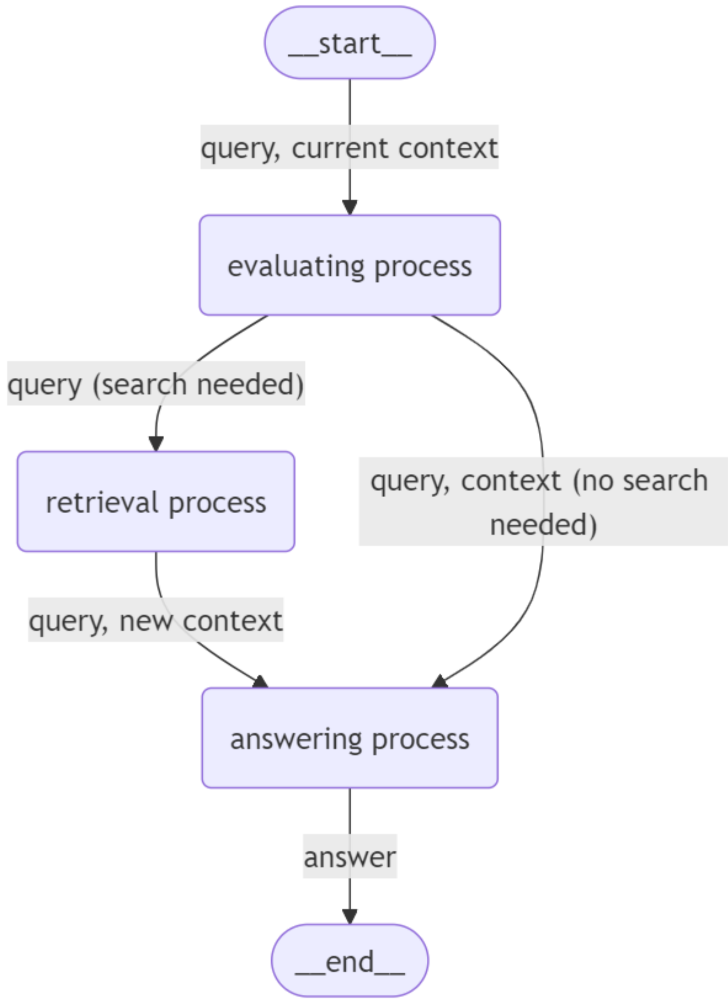



**Figure 4.2:** *The conversation flow*

## 4.3 The Front-End Implementation

### 4.3.1 Interface Design

To provide an accessible and interactive user interface for our retrieval-augmented chatbot, we developed a front-end application using `Streamlit`, a lightweight Python framework tailored for rapid prototyping of web applications. This interface allows users to interact with the chatbot in real time, configure model parameters during development, and inspect the retrieval context when needed.

The application supports two distinct operational modes, catering to both end users and developers:

- **Production Mode**: Initializes the chatbot graph with default parameters, offering a streamlined and user-friendly experience optimized for deployment in real-world scenarios.
- **Debug Mode**: Unlocks advanced configuration options via the sidebar, allowing developers to adjust the large language models used for query expansion and response generation, select between different embedding providers and models and view additional information and metadata regarding document retrieval.

This dual-mode structure ensures that the interface remains both user-friendly and fully configurable, supporting iterative development while preserving accessibility for non-technical users.

### 4.3.2 Chat Interaction

User interaction is facilitated through a chat input field. When a user submits a query, it is appended to the session message history and passed to the backend LangGraph model for processing. The response generated by the graph is then displayed in the chat interface and similarly appended to the history. This architecture supports multi-turn conversations by preserving conversational state across turns via a shared `state` dictionary, which includes both user and assistant messages.

All messages exchanged are displayed using `st.chat_message`, which visually separates user input and assistant responses for readability. A loading spinner is shown while the assistant is generating a response, improving perceived responsiveness.

To enhance transparency in the retrieval-augmented generation process, the interface offers an optional feature that allows users to view the context retrieved during each interaction. After the assistant generates a response, the application checks whether the `context` field in the conversation state contains any documents. If so, the user can click a *Show context* button to display the most recently retrieved documents. For each document, both the source filename and the content are presented.

This feature plays a crucial role in fostering user trust, as it allows users to verify the information provided by the assistant and, when necessary, examine the original phrasing of the content used to generate the response. It also facilitates debugging and supports alignment checking between retrieved sources and final outputs, thereby reinforcing the system's reliability and explainability.


**Figure 4.3:** *The streamlit app*

This Streamlit-based front-end provides a lightweight, modular, and intuitive interface that complements the underlying LangGraph chatbot architecture. Its dual-mode operation enables both end-user deployment and developer experimentation, while its session management, model configurability, and context transparency features collectively contribute to a robust and explainable user experience.

# 5 Chatbot Evaluation

## 5.1 Evaluation Strategy

After having fully developed and implemented our model, the time has come to evaluate it. The evaluation will seek to answer the following key questions:

- How accurately does the system retrieve documents relevant to a user query?
- Does the hybrid approach provide measurable improvements over standalone retrieval methods?
- Are the chatbot's responses consistent with retrieved content and useful from a user perspective?

Unfortunately, evaluating a retrieval-augmented chatbot system is inherently more challenging than evaluating a conventional machine learning model. The open-ended nature of user interactions, the subjective perception of answer quality, and the dual complexity of retrieval and generation make the process less amenable to standardized, purely quantitative metrics. While it is possible to assign qualitative scores to chatbot responses—assessing clarity, tone, or helpfulness—such evaluations are inherently subjective and can vary significantly depending on the evaluator.

To address this, our evaluation strategy is deliberately divided into two independent components, each corresponding to a core task of the system: document retrieval and chatbot response generation. Each component was evaluated separately, using a controlled batch of 100 curated queries.

### 5.1.1 Document Retrieval Evaluation

The first evaluation focused on the system's ability to retrieve the correct document in response to a given user query. To this end, a set of 100 queries was manually constructed, covering a diverse range of difficulty levels, topics, and linguistic formulations, designed to reflect realistic usage scenarios in a corporate wiki. These include:

- **Direct factual queries**, which seek a specific piece of information (e.g., *"I keep receiving emails from APT—what is it?"*).
- **Procedural queries**, asking for step-by-step instructions (e.g., *"How do I create a contact in the CRM?"*).

- **Exploratory queries**, involving broader topics that may require a more comprehensive answer (e.g., *"What tools are used in the onboarding process?"*).

For each query, the corresponding correct document—i.e., the specific wiki page known to contain the answer—was identified in advance. This ground truth enabled a reliable and objective evaluation of whether the system was able to retrieve the relevant content under each retrieval strategy.

Each query was then tested using three retrieval configurations:

1. **TF-IDF (BM25)**: retrieval based on the BM25 ranking function.
2. **Embeddings**: retrieval using embeddings generated by Gemini.
3. **Hybrid**: union of the results obtained by both methods.

Each query was expanded into multiple subqueries using the query expander module powered by an LLM. For each expanded query, both retrieval strategies returned their top 3 documents. The final set of retrieved documents per method was compiled across all subqueries.

The retrieval evaluation used a binary scoring approach: if the correct document appeared among the top 3 results of a method for a given query, it was considered a hit (score 1), otherwise a miss (score 0). This allowed for a simple and interpretable computation of accuracy across methods.

The full evaluation process was implemented in Python and executed through an automated script, which generates a structured JSON file summarizing the retrieval results for each query and method. This output includes the top documents retrieved by TF-IDF, embeddings, and the hybrid strategy, along with the associated query expansions. A second script then processes this JSON file to automatically compute the evaluation metrics, eliminating the need for manual inspection.

### 5.1.2 Chatbot Response Evaluation

The second evaluation phase assessed the quality of the chatbot's responses. A hundred queries were also used, most of them coming from the previous batch (in fact those whose retrieval evaluation was a hit). In this case, the LLM was provided with the context retrieved for the query (with the hybrid method). In order to assess the chatbot's behavior in low-context scenarios, a subset of queries was deliberately tested without any documents provided. This helped evaluate whether the chatbot could correctly identify when it lacked sufficient information and respond appropriately without hallucinating.

Responses were evaluated manually using a 0–5 scale:

- **0**: Factually incorrect or hallucinated response
- **1**: Mostly incorrect or irrelevant; unhelpful

- **2**: Partially correct but incomplete or vague
- **3**: Mostly correct, with minor inaccuracies or gaps
- **4**: Correct and helpful, with clear structure
- **5**: Fully correct, well-structured, and informative

This scoring rubric was applied consistently by the same evaluator to reduce subjectivity and ensure comparability across all queries and configurations.

By decoupling retrieval from generation, this two-pronged evaluation provides a clearer picture of the system's performance: identifying whether failures are due to poor document selection, weak answer synthesis, or both. It allows us to compare retrieval methods independently of generation, and vice versa—offering a more nuanced and actionable understanding of system behavior.

## 5.2 Retrieval Results

The results of the retrieval evaluation are summarized in the following table:

| **Retrieval Method** | **Accuracy** | **Correct / Total** |
|---|---|---|
| TF-IDF (BM25) | 86% | 86 / 100 |
| Embeddings (Gemini) | 90% | 90 / 100 |
| Hybrid (TF-IDF + Embeddings) | **94%** | 94 / 100 |

**Tabla 5.1:** *Retrieval accuracy*

As expected, the hybrid method outperformed the individual methods, demonstrating the complementarity of symbolic (TF-IDF) and semantic (embedding) retrieval strategies. While embeddings performed slightly better than TF-IDF overall, each method retrieved relevant documents that the other missed.

### 5.2.1 TF-IDF and Embeddings

The queries for which each method failed provide insight into their respective strengths and limitations. TF-IDF retrieval, which relies on exact or near-exact term matching, struggled with queries that involved paraphrasing or words that acquire a special meaning when grouped together. For example:

- *"What steps should I follow to open an IT support ticket?"* was missed by TF-IDF but correctly retrieved by embeddings, likely due to embeddings recognizing the broader semantic relation between *"opening a ticket"* and IT support processes.
- *"What is two-factor authentication and what method is recommended to activate it?"* was successfully handled by embeddings, which captured the general concept of two-factor authentication, whereas taken separately, the words *"two"*, *"factor"* and *"authentication"* are too generic and cannot lead to an efficient retrieval by TF-IDF.

Conversely, embedding-based retrieval missed some factually precise but lexically specific queries, which TF-IDF handled well. For instance:

- *"How do I insert a time entry in Toggl?"* — TF-IDF alone retrieved the correct document, as it matched the keywords literally. However, the embedding-based retrieval method failed because the document did not contain any single sentence that clearly described the process. Instead, relevant information was dispersed across multiple paragraphs discussing various topics. This fragmented structure thus weakened the semantic signal for embedding-based retrieval.
- *"I keep receiving emails from APT—what is this?"* was better handled by TF-IDF, which picked up on the acronym *"APT"* and its literal presence in the correct document.

This contrast confirms that TF-IDF excels in matching highly specific vocabulary and exact expressions, while embeddings offer more flexibility in capturing meaning when queries are phrased differently than the source text.

### 5.2.2 Hybrid approach

While the hybrid method performed best overall, it did not succeed in every case. A closer look at the failure cases reveals the limitations and trade-offs of each retrieval strategy.

- *"How do I insert a time entry in Toggl?"* — The hybrid method failed because the embedding-based retrieval method failed completely on this query (for the reasons explained before) and the TF-IDF retrieval was not confident enough on the actual correct document to offset the poor performance of embedding retrieval and thus for the hybrid method to retrieve it.
- *"How do I request vacation days?"* — This was a particularly difficult query. All methods failed to retrieve the correct document, likely because the answer is buried in a single sentence within a long page: *"Holded is the employee portal where you can manage vacation requests, remote work, access payslips, etc."*. The phrasing does not directly answer the query, and the relevant tool (Holded) is only mentioned in passing, making retrieval unlikely.
- *"What kind of file is used to input data into the model, and what information should it contain?"* — This query failed for both TF-IDF and embeddings. The failure likely stems from the vague formulation of the question: it does not specify which model is being referred to, and the terms used (*"file"*, *"input"*, *"information"*) are overly generic. Additionally, the answer is again distributed across a long document, with no paragraph directly addressing the full question.

### 5.2.3 Conclusion

These examples illustrate the strengths and weaknesses of the system. When a query is well-formulated and its answer is clearly and unequivocally expressed within a single document, the system performs reliably. However, challenges arise when:

- The query is ambiguous, vague, or too general.
- The answer is split across multiple sections or deeply embedded in a long document.
- The relevant terms in the document differ significantly from those in the query.

Overall, the hybrid strategy, which merges results from both TF-IDF and embedding-based retrieval, showed the highest overall accuracy. It benefited from the complementary strengths of each method: symbolic precision and semantic understanding. The hybrid method resolved five cases that were missed by TF-IDF alone and two that were missed by embeddings, reinforcing the value of combining both approaches. The hybrid approach significantly improves retrieval robustness, but it remains sensitive to the quality of both the query and the document structure. In conclusion, these results validate the system's design choice to implement hybrid retrieval. They also demonstrate the importance of correct query formulation and, as for any machine learning model (even though this model is not strictly part of the machine learning framework) the quality of input data, namely in this case of the wiki pages style and writing.

## 5.3 Chatbot Response Results

The evaluation of the chatbot's responses yielded promising results, with an average score of 4.3 out of 5 across the test set. Most queries received a perfect score, confirming that the underlying architecture is well designed and that large language models -particularly high-performing ones—are indeed well-suited for projects like this one.

The chatbot proved particularly effective in handling procedural queries, especially those phrased as *"how to"* questions. In such cases, the assistant frequently structured its answers into clear, concise bullet points, offering a direct and actionable response style. Moreover, the responses were generally brief and to the point, often focusing strictly on the information retrieved from the wiki, without attempting to supplement it with external or speculative knowledge. This behavior, while reducing verbosity, strikes a balance between factual precision and completeness.

As seen in the retrieval evaluation, the assistant performed best when the user query was well formulated and its answer clearly available in the provided context. Even when the answer was not present in the retrieved content, the system reliably recognized this and responded accordingly, explicitly stating its limitations and offering a light, contextual filler—typically a fact, joke, or anecdote—to maintain conversational flow. For instance, when asked:

> *"What is the speed of light?"*

providing a page about Toggl (where the answer is obviously not present), the chatbot responded:

> *"I can't answer this question based on the information available in the Toggl documentation. Did you know that the speed of light in a vacuum is approximately 299,792,458 meters per second? That's fast enough to circle the Earth nearly 7.5 times in a single second!"*

This fallback mechanism is an intentional design choice and a clear strength, as it prevents the assistant from hallucinating information while preserving engagement. However, in a few edge cases, the assistant exhibited a slight inconsistency. When the answer was partially present in the document—but not clear or complete—the assistant attempted to provide a partial answer and appended a fallback message (fact or joke) as if the context had been insufficient. An illustrative example is the query:

> *"How do I create a time entry in Toggl for a new client?"*

The relevant page includes technical instructions such as:

> *"The 'client' field must contain the name listed in 'WIP'. If it is not listed, it should be assigned to another generic client called 'Internal'."*

The assistant generated the following response:

> *"To create a time entry in Toggl for a new client, the 'client' field must contain the name listed in 'WIP'. If it is not listed, it should be assigned to another generic client called 'Internal'. Do you know why birds fly south for the winter? Because it's too far to walk!"*

In this case, the chatbot correctly extracted part of the answer but simultaneously triggered the fallback mechanism, suggesting uncertainty about the completeness of the retrieved information. Although this behavior did not result in incorrect facts, it signals a potential refinement point in the logic controlling fallback behavior.

Importantly, during the entire evaluation, the chatbot did not hallucinate any factual content—an outcome that highlights the effectiveness of prompt design and internal safety checks. This reinforces the importance of clear task instructions, well-managed input context, and a robust fallback strategy to ensure high-quality, trustworthy conversational outputs.

# 6 Conclusion

## 6.1 Closing Analysis

### 6.1.1 A Brief Recap

This thesis presented the design, implementation, and evaluation of a retrieval-augmented assistant capable of answering user queries using the content of a corporate internal wiki. The system was developed for Baobab Soluciones with two key objectives in mind. The first was to provide a practical and reliable chatbot interface for internal use, assisting employees in navigating complex documentation. The second—more ambitious—goal was to assess whether such a system could be generalized to customer-facing projects, offering Baobab's clients an intelligent assistant capable of interpreting and querying their own project documentation.

To meet these goals, we gradually arrived at a hybrid retrieval architecture after identifying key limitations in both standalone approaches. Purely embedding-based retrieval struggled with certain types of factual or syntactically similar queries, while TF-IDF lacked semantic understanding and flexibility. By combining both—TF-IDF (specifically BM25) for precise keyword matching, and semantic embeddings for contextual understanding—we sought to balance their respective strengths.

### 6.1.2 A Relative Success

This hybrid strategy proved effective, achieving a retrieval accuracy of 94% on a test set of 100 queries. This result confirms the system's reliability for internal deployment, where an accuracy above 90% is acceptable and occasional retrieval errors can be tolerated. However, the system still falls short of the high precision typically expected for external, customer-facing applications, where errors in document retrieval could undermine trust or lead to misinformation, and where a performance threshold closer to 99% is required.

A modular and graph-based architecture was adopted to support flexible response generation. The assistant could adapt dynamically to different types of queries—answering directly when possible, retrieving knowledge when needed, and gracefully admitting uncertainty when the documentation lacked the answer. Thanks to well-designed control mechanisms and prompt engineering, the assistant avoided hallucinations and maintained user trust throughout.

The response quality evaluation confirmed the validity of this design: the chatbot achieved an average score of 4.3 out of 5 across a broad set of queries. It excelled at "how-to" questions and procedural queries, delivering concise and relevant answers with appropriate formatting. It also performed well in handling unanswerable queries, responding transparently and adding light, engaging content to preserve user experience.

While these results support the feasibility of deploying the system for internal use, they also highlight the gap that remains before it can be reliably used in external client contexts. The main bottleneck is retrieval precision: even a small percentage of missed relevant documents can lead to critical omissions in high-stakes use cases.

In short, this project has demonstrated that building a retrieval-augmented chatbot based on real documentation is not only feasible but effective. The system delivers accurate and context-aware responses in most situations and proves that modern LLMs, when well-guided and properly integrated, are capable tools for document-grounded question answering. However, for the assistant to be deployed in customer-facing projects, further refinement—especially in document retrieval—is still needed.

## 6.2 Retrospective

### 6.2.1 A Complex Reality

Looking back, the development of this project turned out to be significantly more complex than initially anticipated. Retrieval-Augmented Generation is often portrayed in academic literature and online resources as a straightforward and elegant solution: combine the natural language understanding of a large language model (LLM) with the ability to inject up-to-date, domain-specific content via retrieval. The idea is conceptually simple—retrieve the most relevant documents, feed them to the LLM, and let it answer. And indeed, during the chatbot response evaluation, this core idea proved sound: good LLMs are remarkably effective at synthesizing responses when provided with the right information.

However, what initially seemed like a neatly defined system revealed numerous hidden complexities, especially on the retrieval side. The original plan was to rely entirely on semantic search using vector embeddings. Since modern LLMs are supposed to embed meaning beyond surface-level words, this seemed like the best fit for document retrieval. Yet we quickly discovered that embeddings—even those from strong open-source models like DeepSeek—struggled with subtle nuances. Worse, they often failed to align questions with their actual answers. For instance, embeddings would rate two similar-sounding but semantically unrelated questions more highly than a question and its correct answer. These failures were especially frustrating because DeepSeek performs competitively as a chatbot, which led us to suspect that the publicly available embeddings from its API may not be the same ones it uses internally.

When it became clear that the embeddings were unstable and unreliable, we feared this could jeopardize the entire project—after all, traditional RAG systems rely heavily on them. It felt like a dead end.

This challenge led to what was perhaps the most important design pivot in the project: reviving TF-IDF, a more transparent and controllable retrieval method. TF-IDF, especially in its BM25 formulation, may be older, but it is explainable and adaptable. Unlike embeddings, which function as opaque neural representations, TF-IDF vectors are built on interpretable word frequencies. This gave us the flexibility to iterate and improve the system step by step. We enhanced the basic TF-IDF by adding lemmatization, extracting verbs and nouns, performing grammatical conversions between them, fetching synonyms, and leveraging document titles—all tailored to better match user queries. This incremental improvement cycle was one of the most satisfying parts of the project, transforming the initial setback into a productive and creative process.

At the same time, we explored using premium embeddings, notably from Gemini, Google's LLM. Despite having shorter dimensionality, these embeddings proved far more stable and reliable. This finding led to the design of a hybrid retrieval system—one that combines the robustness and customizability of TF-IDF with the semantic depth of modern embeddings.

### 6.2.2 An Optimization Problem

Another particularly challenging aspect of the project was evaluation. Unlike classic machine learning models that come with well-defined benchmarks and metrics, a RAG-based chatbot has to be tested through more qualitative means. There was no standard dataset to test on, no automatic loss function to optimize, and performance was context-dependent. Much of the evaluation had to be done manually: selecting test queries, validating retrieved documents, and rating chatbot answers. This made it difficult to clearly identify weaknesses or quantify improvements—especially when retrieval methods failed silently or for complex reasons.

Moreover, the sheer number of factors influencing performance—prompt design, embedding choice, lemmatization strategy, document chunking, synonym limits—made optimization feel like navigating a vast hyperparameter space without a map. Every improvement introduced new trade-offs, and success often relied on intuition and iteration.

On the other hand, one thing I would do differently if starting again is settling earlier on the tooling. The project went through several restructuring phases: from fully manual file-based retrieval, to custom Python class wrappers, and finally to LangChain's integrated framework. Each refactor brought improvements in usability, but also consumed time and introduced compatibility issues. Had we begun directly with LangChain and its LangGraph framework, we would have had a more stable base from which to build and iterate.

In the end, this retrospective underscores the gap between the apparent simplicity of RAG systems and the depth of engineering required to make them truly reliable. The journey was messier than expected, but also far richer in insights and technical lessons.

## 6.3 Next Steps and Perspectives

While the system developed throughout this project is functional, modular, and already suitable for internal use, there remains significant potential for further improvement and expansion. Several promising directions can be pursued to enhance its accuracy, usability, and adaptability to new use cases and deployment contexts.

### 6.3.1 System Evaluation

One of the main challenges during development was the lack of automated, scalable evaluation metrics. In future work, we propose building a standardized benchmark dataset of queries with known answers, ideally labeled with correct documents and ideal responses. This would allow automated evaluation of both retrieval and response generation, enabling faster iteration and fairer comparison of different models and configurations.

Another important avenue is the incorporation of user feedback. Currently, the system operates in a static mode—user interactions do not influence future performance. Introducing feedback loops that collect user ratings or corrections could allow for adaptive tuning of retrieval parameters, model prompts, or even query reformulations over time. This would gradually improve accuracy and help prevent recurring mistakes in both retrieval and generation.

### 6.3.2 Document retrieval

On the retrieval side, although the adoption of Gemini embeddings already marked a significant improvement over earlier models, our system still relies on static similarity between query and document vectors. This means that when a query is issued, it is embedded and compared to the stored document embeddings based solely on vector similarity (e.g., cosine distance). While efficient, this method does not capture the detailed interaction between the query and each candidate document. A promising future improvement would be to introduce a re-ranking step after the initial retrieval. This involves using a cross-encoder model that takes both the query and each retrieved document as joint input and evaluates how well the document truly answers the query. This would help the system better distinguish between documents that are superficially similar and those that are genuinely relevant, resulting in more accurate rankings and more reliable responses.

The quality of retrieval could also be improved by a better preprocessing of the user queries. An LLM could be used to interpret and transform the query before passing it through the pipeline. For instance, if the user enters a compound query (one that implicitly contains two or more separate questions), the system could split it into subqueries, retrieve documents independently for each, and then synthesize a coherent, multi-part answer. Similarly, the LLM could detect ambiguities, rephrase vague queries into more precise ones, or inject missing keywords that are contextually implied. These refinements would improve both retrieval relevance and the final response quality.

Another direction involves expanding the system's multilingual capabilities. Although we currently translate all inputs to Spanish to match the wiki content, this limits flexibility. A more robust solution would allow documents and queries in multiple languages, performing cross-lingual retrieval and enabling use in international or multilingual environments. This would be particularly beneficial for client-facing deployments in companies operating across different regions.

### 6.3.3 Chatbot Memory

Finally, from a user experience perspective, improving the system's handling of memory within a conversation could significantly enhance coherence and contextual awareness. Currently, while basic state is maintained, the assistant does not fully leverage the evolving context of an ongoing dialogue—such as previously asked questions, intermediate clarifications, or references to earlier answers. Introducing a more robust short-term memory module would allow the chatbot to track and integrate key elements from earlier in the same conversation. This would enable more fluid follow-ups, reduce repetition, and support complex, multi-step interactions where information accumulates progressively.

In summary, while this project has produced a solid foundation for a retrieval-augmented assistant tailored to internal wiki content, its architecture leaves ample room for improvement. With deeper evaluation, smarter query handling, and performance optimizations, the system could evolve into a robust platform suitable for both internal documentation support and external project integration.

# Bibliography


[1] Fareed Khan. *All RAG Techniques: A Simpler, Hands-On Approach*. 2025. URL: https://github.com/FareedKhan-dev/all-rag-techniques.

[2] Nir Diamant. *Advanced RAG Techniques: Elevating Your Retrieval-Augmented Generation Systems*. 2024. URL: https://github.com/NirDiamant/RAG_Techniques.

[3] Nir Diamant. *GenAI Agents: Comprehensive Repository for Development and Implementation*. 2024. URL: https://github.com/NirDiamant/GenAI_Agents.

[4] Akiko Aizawa. "An information-theoretic perspective of tf–idf measures". In: *Information Processing & Management* 39.1 (2003), pp. 45–65. URL: https://www.sciencedirect.com/science/article/pii/S0306457302000213.

[5] Tarik Moufakir. *Comparaison de TF-IDF et BM25 pour le repérage de l'information*. Université du Québec à Montréal, 2014. URL: https://archipel.uqam.ca/7781/.

[6] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks". In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin. Vol. 33. Curran Associates, Inc., 2020, pp. 9459–9474. URL: https://proceedings.neurips.cc/paper_files/paper/2020/file/6b493230205f780e1bc26945df7481e5-Paper.pdf.

[7] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yixin Dai, Jiawei Sun, Haofen Wang, and Haofen Wang. "Retrieval-augmented generation for large language models: A survey". In: *arXiv preprint arXiv:2312.10997* 2.1 (2023). URL: https://simg.baai.ac.cn/paperfile/25a43194-c74c-4cd3-b60f-0a1f27f8b8af.pdf.

[8] Sumit Kumar Dam, Choong Seon Hong, Yu Qiao, and Chaoning Zhang. *A complete survey on llm-based ai chatbots*. 2024. URL: https://arxiv.org/abs/2406.16937.

# A

# The System Layout

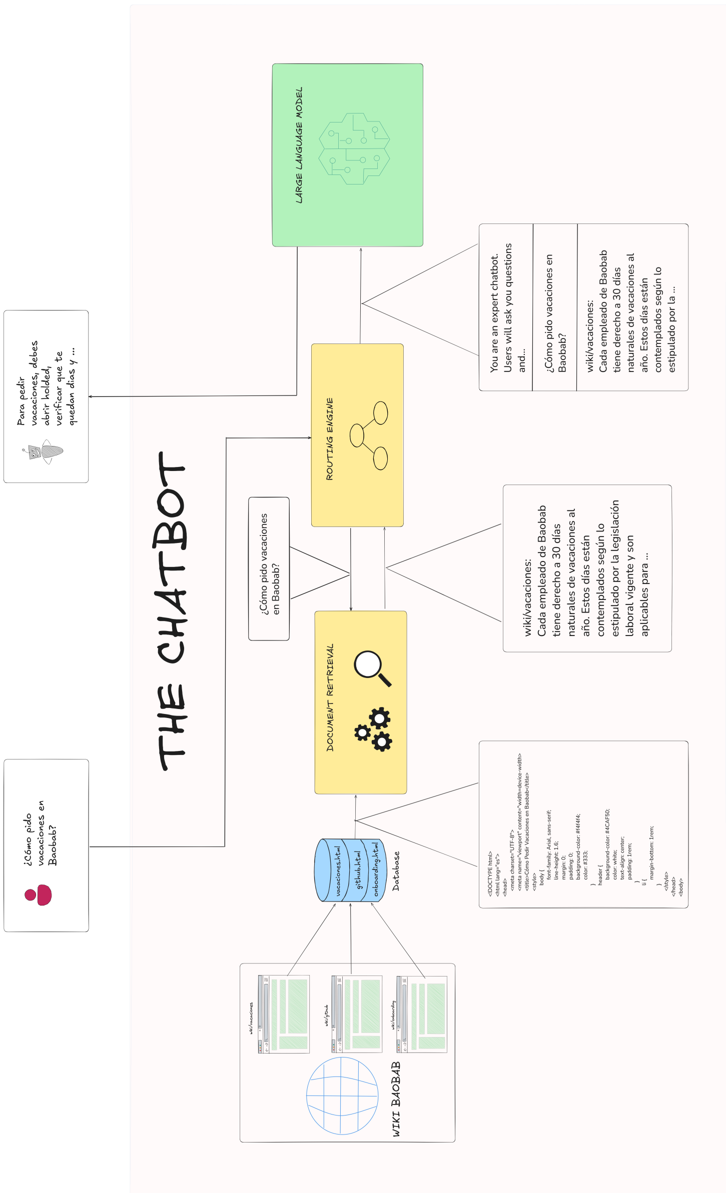



**Figure A.1:** *An illustration of the whole system*

UNIVERSIDAD PONTIFICIA
COMILLAS

WIKI LLM: DESIGNING A
RELIABLE ASSISTANT CHATBOT
FOR EFFICIENT QUERIES IN A
CORPORATE WIKI

GUILLAUME GUERS

MADRID, JUNE 2025