



**POLITECNICO**  
**MILANO 1863**

SCUOLA DI INGEGNERIA INDUSTRIALE  
E DELL'INFORMAZIONE

# Generative AI for Customer Support:

## An Empirical Investigation of RAG-Based Chatbot Architectures in an ERP Scenario

TESI DI LAUREA MAGISTRALE IN  
COMPUTER SCIENCE AND ENGINEERING -  
INGEGNERIA INFORMATICA

Author: **Bianca Christiana Savoiu Marinas**

Student ID: 221629

Advisor: Prof. Mark James Carman

Co-advisors: Michele Silvestri, Gaetano Palmieri, Simone Men, Nicolò Brunello

Academic Year: 2024-25



# Abstract

The thesis investigates the development of a customer support chatbot in an Enterprise Resource Planning (ERP) system to reduce help desk workload and enhance user satisfaction. The main objectives are to decrease support tickets by 20%, achieve a user satisfaction rate of 80% or higher, and minimize human error in ticket creation, while ensuring high performance and efficiency.

The solution development process is based on a progressive, data-driven methodology, and it employs off-the-shelf Large Language Models (LLMs), but given that these models are not trained on company-specific data, they need to be augmented with other techniques, such as Retrieval-Augmented Generation (RAG) and Prompting, to effectively integrate the knowledge base and tailor the model to the task. The selected LLMs leverage extensive pretraining on diverse datasets, providing a strong understanding of natural language, including Italian, which avoids the need for resource-intensive model training from scratch. Instead, the focus is on combining these models with the use of external private knowledge and carefully crafted instructions to achieve the desired quality of the chatbot; thus, this thesis is based on the development of a RAG.

The evaluation of the efficacy of the chatbot posed a great challenge in the absence of a real evaluation dataset. A part of this work is focused on creating an evaluation framework, more precisely a pipeline to synthetically generate an evaluation set, using LLMs inference and LLM as a judge. The results will be evaluated using a set of carefully selected metrics (ROUGE-1, BERT-F1 score, and SemScore), applied to the synthetically generated dataset. These metrics, alongside human feedback will assess the quality of the chatbot's responses, to identify the optimal RAG-based chatbot solution.

This thesis contributes to ongoing research on RAG-based chatbot assistants, balancing performance and implementation complexity, iteratively improving the system, and evaluating its cost-effectiveness trade-off, using both automatic metrics and human feedback.

**Keywords:** ERP, LLM, RAG, chatbot, customer support, synthetic evaluation set



# Abstract in Lingua Italiana

Questa tesi esplora lo sviluppo di un chatbot per il supporto clienti in un sistema ERP, con l'obiettivo di ridurre il carico di lavoro dell'help desk e migliorare la soddisfazione degli utenti. Gli obiettivi principali sono: ridurre la quantità di ticket del 20%, ottenere un tasso di soddisfazione maggiore del 80% e minimizzare gli errori umani durante la creazione dei ticket, garantendo allo stesso tempo alte prestazioni ed efficienza.

Il sistema si basa su una metodologia progressiva e guidata dai dati, utilizzando LLM preaddestrati. Poiché questi modelli non sono allenati sui specifici dati aziendali, i modelli devono essere integrati con altre tecniche, come Retrieval-Augmented Generation (RAG) e Prompting per aggiungere la base dati e adattare il sistema al compito desiderato. I LLM selezionati sfruttano un pretraining su dataset diversificati, garantendo una solida comprensione del linguaggio naturale, incluso l'italiano. Questo approccio evita addestramenti costosi, concentrandosi sulla combinazione di conoscenze esterne e istruzioni mirate con le capacità dei modelli, per migliorare la qualità del chatbot, quindi questa tesi si basa sullo sviluppo di un sistema RAG.

La valutazione del chatbot è complessa in assenza di un dataset reale. Per questo, è stata sviluppata una pipeline di valutazione sintetica, sfruttando LLM per la generazione e come giudici. I risultati vengono analizzati con metriche accuratamente selezionate (ROUGE-1, BERT-F1 e SemScore), applicate a un dataset generato sinteticamente. Queste metriche, insieme al feedback umano, valuteranno la qualità delle risposte per identificare la soluzione RAG ottimale.

Questa tesi contribuisce alla ricerca sui chatbot RAG, bilanciando prestazioni e complessità implementativa, migliorando iterativamente il sistema e valutando il compromesso tra costo ed efficacia, tramite metriche automatiche e feedback umano.

**Parole chiave:** ERP, LLM, RAG, chatbot, assistenza clienti, dataset di valutazione sintetico



# Contents

|                                                                                   |            |
|-----------------------------------------------------------------------------------|------------|
| <b>Abstract</b>                                                                   | <b>i</b>   |
| <b>Abstract in Lingua Italiana</b>                                                | <b>iii</b> |
| <b>Contents</b>                                                                   | <b>v</b>   |
| <br>                                                                              |            |
| <b>Introduction</b>                                                               | <b>1</b>   |
| <br>                                                                              |            |
| <b>1 Introduction to ERP Systems and Customer Support Challenges</b>              | <b>3</b>   |
| 1.1 ERP systems and the need for customer support . . . . .                       | 3          |
| 1.2 The importance of customer support automation and conversational agents       | 4          |
| 1.3 The contribution of this research to the field of conversational agents . . . | 5          |
| <br>                                                                              |            |
| <b>2 Theoretical Foundations</b>                                                  | <b>7</b>   |
| 2.1 Natural Language Processing . . . . .                                         | 7          |
| 2.2 Machine Learning and Neural Networks . . . . .                                | 8          |
| 2.3 Large Language Models and Chatbots . . . . .                                  | 9          |
| 2.4 Question Answering and Retrieval Augmented Generation . . . . .               | 11         |
| 2.4.1 Information Retrieval and Question Answering . . . . .                      | 12         |
| 2.4.2 Retrieval Augmented Generation . . . . .                                    | 12         |
| <br>                                                                              |            |
| <b>3 Retrieval-Augmented Generation (RAG) Advanced Techniques</b>                 | <b>17</b>  |
| 3.1 Advanced indexing . . . . .                                                   | 17         |
| 3.2 Advanced retrieval . . . . .                                                  | 18         |
| 3.3 Advanced generation . . . . .                                                 | 19         |
| <br>                                                                              |            |
| <b>4 Proposed Methodology and Implementation</b>                                  | <b>23</b>  |
| 4.1 Requirements and implementation strategy . . . . .                            | 23         |
| 4.1.1 Requirements analysis and extended use case . . . . .                       | 23         |
| 4.1.2 Implementation strategy . . . . .                                           | 26         |

|          |                                                      |            |
|----------|------------------------------------------------------|------------|
| 4.2      | Data creation and preprocessing . . . . .            | 27         |
| 4.2.1    | Corpus composition . . . . .                         | 27         |
| 4.2.2    | Augmented Dataset Creation . . . . .                 | 28         |
| 4.2.3    | Data preprocessing experiments . . . . .             | 33         |
| 4.2.4    | Chunking strategies . . . . .                        | 39         |
| 4.2.5    | Embedding model and Vector-store creation . . . . .  | 48         |
| 4.3      | Naive RAG implementation with improvements . . . . . | 50         |
| 4.3.1    | Baseline Naive RAG implementation . . . . .          | 50         |
| 4.3.2    | Improved Naive RAG implementation . . . . .          | 51         |
| 4.4      | Advanced RAG implementation . . . . .                | 58         |
| 4.4.1    | Query rewriting and Reciprocal Rank Fusion . . . . . | 59         |
| 4.4.2    | Comparison of LLM generation . . . . .               | 65         |
| <b>5</b> | <b>Evaluation Metrics and Testing Framework</b>      | <b>67</b>  |
| 5.1      | Testing Framework and Evaluation Set . . . . .       | 67         |
| 5.2      | Evaluation Metrics . . . . .                         | 73         |
| 5.2.1    | Research questions on LLM juries . . . . .           | 76         |
| <b>6</b> | <b>Conclusions and Future Developments</b>           | <b>81</b>  |
|          | <b>Bibliography</b>                                  | <b>85</b>  |
| <b>A</b> | <b>Appendix A</b>                                    | <b>89</b>  |
| <b>B</b> | <b>Appendix B</b>                                    | <b>91</b>  |
|          | <b>List of Figures</b>                               | <b>113</b> |
|          | <b>List of Tables</b>                                | <b>117</b> |

# Introduction

Customer support plays a critical role in the overall user experience in any domain and particularly for businesses relying on complex systems such as Enterprise Resource Planning (ERP) software. The ERP is a business management software, essential for managing the core operations of the company, such as accounting, supply chain, and human resources. Given the inherent complexity of these activities and of the software itself, users frequently encounter issues they cannot solve independently, leading to a high volume of ticket creation for support requests. This significant volume of requests places a heavy burden on the customer support teams, the so called help desk, causing delays in response times for a lot of customers, and a potential decrease in user satisfaction.

In many cases, when using the ERP software, users need immediate solutions for issues such as system errors, functionality misconfigurations, or procedural questions on how to accomplish some tasks. However, the traditional support model where users log support tickets and wait for human assistance can be slow and inefficient. As ticket volumes grow, support teams struggle to address all issues in time, causing bottlenecks in resolving problems and reducing overall efficiency. This situation not only impacts the user experience, but also increases operational costs and the likelihood of human errors in ticket handling.

To address these challenges, there is a growing interest in leveraging artificial intelligence (AI) techniques and conversational agents to automate parts of the support process. Specifically, Retrieval-Augmented Generation (RAG) based chatbots can assist users by offering immediate responses based on external knowledge, such as private technical documentation. The goal is to have the chatbot help the customers, as much as possible, to resolve their issues without the need for human intervention. Then, when the chatbot knowledge is not enough and human support becomes indispensable, the system can also help the user to send a better support request, automatically generating a well-structured ticket for the support team, reducing human error and enhancing the accuracy of ticket creation.

The goal of this thesis is to explore how RAG-based chatbots can be implemented in an ERP context to effectively reduce the number of support tickets by solving user problems

in real-time. By integrating a Large Language Model (LLM) with the ERP system's documentation, implementing a RAG solution, the chatbot will serve as the first point of contact for users, offering an intelligent solution to their queries. This approach has the potential to transform customer support for ERP systems, ensuring faster issue resolution, lower ticket volumes, and an enhanced user experience, while simultaneously reducing costs.

A scalable approach to RAG-based chatbot development will be presented, starting from a simple RAG pipeline with off-the shelf models and advancing to more sophisticated methods as necessary. The chapters will be organized as follows:

1. **Chapter 1:** explains the ERP context and the customer support challenges, showing the rationale behind the RAG-based chatbot development approach;
2. **Chapter 2:** describes the core theories and tools in Natural Language Processing (NLP), Machine Learning (ML), Large Language Models (LLMs) and Retrieval-Augmented Generation (RAG), that underpin the models utilized in this work;
3. **Chapter 3:** describes the possible improvements of a basic RAG system, integrating more advanced techniques for better performance, which are the focus of the implemented RAG-based chatbot solutions;
4. **Chapter 4:** presents an use case to better understand the task at hand and also showcases the methodological and implementation choices underlying the proposed chatbot solution, presenting some evaluation results, along with a discussion of the final outcomes achieved;
5. **Chapter 5:** defines the testing framework and the evaluation metrics used to assess the chatbot's performance, while explaining the challenges associated with the system evaluation;
6. **Chapter 6:** summarizes the key achievements of this thesis, identifies its limitations, and outlines potential future developments.

# 1 | Introduction to ERP Systems and Customer Support Challenges

## 1.1. ERP systems and the need for customer support

Enterprise Resource Planning (ERP) is a category of business management software, used to manage day-to-day business activities such as accounting, project management, risk management and supply chain operations. The ERP system covers many functional areas of a business, such as financial accounting, management accounting, human resources, manufacturing, order preprocessing, project management and so on. The ERP manages the integration of these different business processes and it is composed of a suite of integrated applications that an organization can use to collect, store, manage and interpret data from many business activities, controlling the flow of data between processes eliminating data duplication and providing data integrity [1].

Today, ERP systems are critical for managing thousands of businesses of all sizes and in all industries and they can be deployed on-premise or in the cloud. The core values of these systems lie in the ability to improve collaboration and decision-making through real-time insights of heterogeneous business processes and their interaction, using common databases maintained by a database management system. The most fundamental advantage of ERP is that the integration of a myriad of business processes saves time and expense, centralizing the business data. Simply put, ERP is the vehicle for integrating people, processes and technologies across modern enterprise [2].

ERP systems provide numerous benefits, such as improved efficiency, robust data security, greater organizational agility and flexibility, as well as enhanced collaboration and decision-making capabilities. However, their inherent complexity can pose substantial challenges for users and this is why customer support becomes crucial in ERP systems in order to avoid the disruption of the business operations. Customer support addresses a

range of different issues encountered by the user, including system errors, misconfigurations, data inconsistencies and procedural ambiguities. These challenges result in a high volume of support requests in the form of tickets, which the help desk teams struggle to handle without delays, reducing user satisfaction and generating inefficiencies in the business operations. In this context the automation of customer support becomes crucial, in order to maintain the ERP benefits, offering immediate and precise assistance to the users, guaranteeing high efficiency, real-time responses and cost savings by reducing dependence on human agents. This automation is in accordance with the objectives of the ERP systems, which aim to enable users to focus on strategic matters and streamline business operations, without time delays and operational impediments.

The main challenges in an ERP system help-desk are [5]:

1. Slow response times, given that there are different priorities and the issues can require time to be diagnosed and fixed;
2. Answering challenging questions;
3. Frequent transfers to different agents and teams, causing delays and miscommunication, impacting negatively the customer service;
4. Handling multiple customers at the same time;
5. High query volumes that lead to ticket backlogs, the accumulation of unresolved tickets.

## **1.2. The importance of customer support automation and conversational agents**

The implementation of conversational agents represents the first step toward automating customer support. Conversational agents, typically referred to as chatbots, are a type of software that can simulate a conversation with a user in natural language. The conversational agents can be classified in two categories, Rule-Based Agents and AI-Powered Agents, that will be explained in more detail in the following chapter. The work of this thesis focuses on AI-Powered Conversational Agents, which enable efficient customer assistance in the use of the ERP software, providing 24/7 support by answering customer queries, resolving issues and guiding the users through various processes and procedures [4]. These agents can perform the equivalent work of numerous humans supporting many customers at the same time, thus their introduction saves operational costs for organizations while minimizing delays and reducing ticket creation. The conversational agent

communicates in natural language with the user trying to answer in a clear and helpful way to the user's problems and needs. A next step, when the agent is not able to solve the problem, is to guide the user to the ticket creation then assigned to available human agents in order to have human intervention and personalization when needed. As already anticipated, the thesis work will focus mainly on the first part of this automation, in which the AI chatbot will need to understand the context, the user need and retrieve the correct solution to answer appropriately. Thus, with the customer service automation in ERP support through conversational agents, there are some important benefits:

1. 24/7 availability and reduced delays in customer support;
2. Enhanced efficiency and productivity of the customer service teams, which can focus more on more complex problems, while the automatic agents solve repetitive and simpler user needs;
3. Minimized ticket backlogs, reducing the accumulation of unresolved tickets and improving customer satisfaction.

However, even if the automation offers considerable benefits, it also has its limitations, such as loss of personal touch, limited understanding of nuanced queries, and the inability to handle complex or novel problems that may require human creativity and critical thinking [5]. To address these challenges, this thesis proposes a hybrid approach: utilizing a chatbot for initial interactions, such as user doubts, routine problem resolution and assistance with consolidated processes and procedures of the ERP software, while for more complex problems it is necessary to generate a support ticket, ensuring that expert human agents provide specialized assistance. This approach enhances the quality of customer service, while mitigating the inherent risks of the used technology.

### 1.3. The contribution of this research to the field of conversational agents

In this thesis, we explore mainly the capabilities of a conversational agent augmented with technical documentation knowledge on the various business processes and procedures of the specific ERP software, **Panthera**. The focus is on an AI chatbot that integrates Retrieval-Augmented Generation (RAG) techniques in order to assist users with issues and doubts resolution in real-time with up-to-date knowledge, reducing the workload on the help desk teams. RAG is a method that combines Information Retrieval (IR) techniques to retrieve documents that are likely to have information that might help answer the user's question, and Large Language Models (LLMs) used to generate the

answer given the extracted documents. Although the current implementation focuses on guiding users and delivering accurate information, future developments of the system aim to expand its capabilities to include automated ticket creation.

This research adopts a cost-effective iterative strategy, progressing from simpler solutions to more advanced and expensive ones. Analyzing the trade-off between cost and performance is decided the optimal stopping point across different implementation approaches, in order to achieve high user satisfaction at an acceptable cost. The study is focused on different aspects of the implementation of a RAG-based chatbot and its evaluation:

1. Exploration of the potential of basic and advanced RAG techniques, analyzing how RAG-based chatbots can improve the quality of the model responses in the context of the ERP customer support, balancing cost, complexity and performance;
2. Addressing evaluation challenges in a scenario in which there is not a clearly defined test set, thus, a significant part of this work involves creating a synthetic dataset using advanced data generation techniques, in order to have a robust and reliable evaluation at each stage of the implementation process;
3. Development of an evaluation framework, investigating the most accurate evaluation metrics for assessing responses in the specific customer support context, with the goal of selecting the metrics that better align with human feedback and expectations, enabling an automated evaluation of the system performance;
4. Identification of possible future developments to improve the chatbot system and extend its functionality to support automated ticket creation.

By focusing on these objectives, this thesis contributes to the growing body of research on AI-driven customer support, with a particular emphasis on RAG-based conversational agents in the context of ERP systems. The proposed chatbot aims to reduce operational costs and relieve workload on support teams, by decreasing the volume of support tickets by at least 20%, while significantly improving overall efficiency and user satisfaction, achieving a user satisfaction rate of 80% or higher.

## 2 | Theoretical Foundations

This chapter introduces the theoretical concepts and some technologies of Natural Language Processing (NLP), Artificial Intelligence (AI), Large Language Models (LLM) and Information Retrieval (IR), which represent the core ideas behind this thesis. These concepts are essential for comprehending the Retrieval-Augmented Generation (RAG) techniques and the RAG-based chatbot implementation discussed in the following chapters.

### 2.1. Natural Language Processing

Natural Language Processing (NLP) is a subfield of computer science and AI that uses Machine Learning to enable computers to understand and communicate with human language. NLP combines computational linguistics with other fields like statistical modeling, machine learning and deep learning, enabling systems to understand and generate text and speech. The goal of this field is to fill the gap between human communication and computer understanding, giving machines the ability of interacting with humans in a natural and efficient way.

NLP is an engineering discipline that aims to build technology to accomplish the automation of useful tasks that concern the computational analysis, interpretation, and production of natural language, in either written or spoken language. It can be divided in two somewhat overlapping sub-fields: natural language understanding (NLU), which focuses on understanding the intended meaning of a text, and natural language generation (NLG), which focuses on text generation by means of a machine. [7] NLP encompasses a wide range of activities, including the following activities, important for the work presented in this thesis:

1. Text Analysis: extracting meaning and useful information from texts, such as in topic identification and information extraction;
2. Text Generation: generating human-readable text, text summarization and dialogue generation;

3. Linguistic Interaction: enabling computers to interact with humans using natural language, such as chatbots, virtual assistants and question-and-answer systems;
4. Information Retrieval: finding the documents from a collection that are most relevant to a question.

It is important to consider that human language is highly expressive and often ambiguous, capable of conveying profound meaning while also being redundant. These inherent complexities make NLP a particularly challenging field. However, the integration of NLP with machine learning and deep learning techniques has revolutionized the field and the activities explained before, enhancing the performance of these language-related tasks over time.

NLP is also, one of the main foundations of generative AI, researching the communication skills of LLMs and the impact of using linguistically structured natural language prompts for clearer commands and better overall results. Due to the contributions of NLP to these cutting-edge techniques, it plays an important role in automating business operations and enhancing productivity, particularly in applications like customer support.[6]

## 2.2. Machine Learning and Neural Networks

Machine Learning is a subfield of Artificial Intelligence concerned with the study and development of statistical algorithms that can enable computers to learn patterns from data and generalize to unseen data. This allows computers to perform certain tasks without requiring explicit programming [12]. Machine Learning finds application in many fields, including NLP, in which it plays a crucial role for tasks such as text classification, sequence labeling and language modeling. Among these tasks, the most relevant for this thesis is language modeling, which involves predicting the likelihood of a sequence of words, which is the fundamental component of text generation, used at the core of chatbots.

Neural Networks (NNs) are a class of machine learning algorithms, originally inspired by the structure and functionality of biological neural networks in animal brains. Artificial neural networks (ANNs) consist of interconnected nodes, called neurons, organized into layers. Each neuron, in a layer, receives an input, processes it with a mathematical function, and produces an output [13]. Theoretically, even a minimal neural network, with a single hidden layer, can learn any function, so these networks are a powerful tool, being able to learn complex pattern in data and perform various tasks. This makes NNs suitable for NLP tasks, and common types of NNs used in NLP are:

1. **Feedforward Networks:** Are multilayer networks in which the units are connected with no cycle, so they process information in a single direction, from input to output layers, and no outputs are passed back to the lower layer; these are often used in simple classification tasks;
2. **Recurrent Neural Networks (RNNs):** These incorporate feedback loops, allowing them to process sequential data, such as text, and to capture temporal dependencies, which are crucial given that language is a temporal phenomenon, having a sequential nature. The RNNs are extremely effective when applied to language, given that they allow to represent prior context, allowing the model's decisions to depend on many words in the past. These networks can also be used to generate text, making them useful for generative AI tasks;
3. **Long Short-Term Memory Networks (LSTMs):** A variant of RNNs designed to handle long-range dependencies, improving performance on tasks that require the retention of information over many steps. This variant was necessary, because RNNs are difficult to train for tasks that require the network to make use of information distant from the current point, and in language applications and generative AI tasks, it can be crucial to take into consideration distant information in the text;
4. **Transformers:** A neural architecture that uses attention mechanisms to capture relationships between words in a sequence, achieving state-of-the-art performance in many NLP tasks.

The use of modern NNs is called Deep Learning, because modern networks are deep, having many layers of neurons. In modern applications the go-to technology is based on transformers, which is now the standard architecture for building Large Language Models and consequently chatbots.

## 2.3. Large Language Models and Chatbots

Large Language Models (LLMs) are advanced machine learning models, based on transformers and designed for NLP tasks. The LLMs are language models with many parameters and they are trained with self-supervised learning on a massive amount of text. [14]. The pretrained LLMs have remarkable performances on NLP tasks thanks to the knowledge learned during pretraining, and these models have been especially significant for their results in applications, such as question-answering and chatbots, making them the fundamental models of the approaches explored in the specific task of this thesis. LLMs can be fine-tuned for specific tasks or guided by prompt engineering to enhance the per-

formance and tailor their behavior to specific requirements. Since fine-tuning is both very expensive, especially when applied to such big models, and also reliant on high-quality, realistic and curated data, this thesis will focus on prompting as the primary approach for further model learning, more specifically in-context learning, employing an iterative process of prompt engineering with human feedback, as detailed in the following chapters.

The task of the LLM in this application is that of conditional generation, the task of generating text token by token, having the text conditioned on the prompt, which is the input piece of text that gives the model context and specific instructions on the task. In this way, using the prompt, the LLM can perform a task it wasn't actually trained for, simply by leveraging its understanding of human language and thus, the instructions provided in the prompt as text. This ability, known as generalization capability, is an important characteristic of LLMs: the extensive linguistic knowledge allows LLMs to generalize well to new tasks and domains. However, this capability alone is not enough to achieve good performance, especially in applications like this one, where the chatbot assistant must generate answers based on private documentation that the model has never encountered during pretraining and has no prior knowledge of.

As mentioned before, to obtain a pretrained LLM is necessary to train a transformer as a language model, using self-supervised learning. The process consists in feeding to the transformer a corpus of text data asking the model to predict at each time  $t$  the next word. The model is self-supervised because the training material doesn't need any special label, having the natural sequence of words as its own supervision. The model is trained to minimize the error in predicting the true next word in the training sequence, using cross-entropy as the loss function [8]. It is worth noticing that LLMs are typically trained on data scraped from the web, textbooks, and other curated datasets. However, not all training data is of high quality, and biases or inaccuracies may be introduced during training. Furthermore, the data available to the model are not always up to date, because this would require a continuous training on new and updated data, which is expensive. Although the enormous pretraining data of LLMs contains many domains, it is often the case, as in this thesis, that the model needs to be applied to a new domain or even to a new task that may have appeared only sparsely in the pretraining, or not at all. In this thesis, the private documentation of the company is an example of such a domain. Since the model has no prior exposure to this data, one approach could be to fine-tune the model by continuing its training on the company's relevant documents. However, fine-tuning requires significant computational resources and time, which were not feasible given the limited resources at disposal for this project. As a result, alternative methods were chosen, primarily Retrieval-Augmented Generation (RAG) and prompt engineering.

LLMs are at the base of chatbots, which are systems designed to simulate conversations with users. There are also rule-based chatbots, that will not be discussed here, focusing instead on the machine-learning based ones. These chatbots generate responses from a dataset of conversations and text, giving support to the customers, answering to their queries with the knowledge acquired from existent pretraining and from the private documentation, reducing in this way the workload on humans agents from the help-desk unit of the company. Prompt engineering, as aforementioned, is a key technique for optimizing the chatbot performance and it involves crafting a text input to guide the LLM towards generating the desired output. In the prompt, there can be included information relevant to the task, additional context retrieved in some way from external data, specifics about the preferred structure of the output, and also positive and negative examples demonstrating the task and guiding the model.

Despite the achievements of LLMs in many NLP tasks, these models also have many potential problems, which become realized when pretrained LLMs are used for downstream tasks, particularly for tasks involving text generation and question answering. The main and more common problem of LLMs and therefore LLM-based chatbots are the hallucinations, the fact that the models can output text that contains false information. This happens because the models are trained to generate text that is predictable and coherent given the training seen so far, but it doesn't enforce the generated text to be correct or true, just predictable. In the application addressed in this thesis, the reliance on private company documentation further emphasizes the need for a robust method. To address these challenges, the proposed approach is based on Retrieval-Augmented Generation, combining the strengths of LLMs with retrieval mechanisms to improve accuracy and relevance, minimizing as much as possible hallucinations and undesired behaviors [8].

## 2.4. Question Answering and Retrieval Augmented Generation

Retrieval-Augmented Generation (RAG) is a hybrid approach that combines the generative capabilities of LLMs with the precision of information retrieval systems. In a RAG system, an information retrieval component fetches relevant documents from a database based on a query or prompt. The LLM uses the prompt, together with the extracted documents, as context to generate accurate, informed responses. RAG systems proved to be particularly effective for question answering tasks, being a way to provide fact-based updated answers, by incorporating information from the retrieved relevant parts of documents.

### 2.4.1. Information Retrieval and Question Answering

Information Retrieval (IR) aims to identify the most relevant documents within a collection that best address a given question. Document retrieval systems consist of two processes: indexing and matching. The indexing can be done using a vector store model, while the matching part is done using similarity or distance scores.

Question answering is the task of answering questions posed by humans in a natural language and can be of two types: multiple choice and open domain. For customer support, the chatbot needs to deal with an open-domain question-answering problem, so the model has to provide answers without any provided options. The system needs to answer to human information needs, which in this context has to be precise and accurate in order to guide the user in the ERP system. Given the problems of LLMs of hallucinations and the fact that simply prompting a LLM is not enough in order to ask questions about proprietary data, the most common way to tackle a question-answering problem in this context with LLM is by using Retrieval Augmented Generation. In this technique, IR is an essential step to retrieve the necessary information from the company's internal documentation that contains the answers for customer service. While simple LLMs have also problems with questions about rapidly changing information, given that LLMs won't have updated information from after their release, using IR in a RAG system allows to retrieve the updated information from the documents and then generate the correct and up-to-date answer for the customer. Basing the generated answers on retrieved documents, the output is grounded in facts from curated datasets, reducing the hallucinations of the model and resulting in more robust and accurate systems [8].

### 2.4.2. Retrieval Augmented Generation

The RAG system can be defined as a model composed of a retriever and a reader. The retriever uses IR techniques, extracting relevant parts of documents from a text corpus, using different retrievers and search techniques to find the relevant information among all the stored data. In the second stage of the RAG, there is a reader, that takes as input a prompt and the relevant information extracted in the first step, using all this context and instructions to generate an answer token by token. Using this idea of generating the answer with an LLM, conditioning the generation on the retrieved passages and some additional prompt, it is possible to build a more reliable chatbot, a system that can carry on extended robust conversations, basing its knowledge on the available documentation. Thus, the RAG enhances the LLMs capabilities, retrieving relevant document chunks from external knowledge, reducing the generation of incorrect content, making the RAG

one of the main technologies to enhance chatbot performance in real-world applications at a reasonable cost.

The basic RAG architecture, called Naive RAG, includes three main operations [15]:

1. Indexing: the data is converted in text format, and given the input limitations of language models, the text is segmented into smaller pieces, called chunks, which are encoded in vector representations using an embedding model and are stored in a vector database, from which the relevant information will be extracted by similarity search in the subsequent phase;
2. Retrieval: given a user query, the system uses the same encoding employed during the indexing, transforming the query in a vector representation, used to compute a similarity score between the query vector and the vector of chunks in the database, retrieving the top  $k$  chunks, the ones that are most similar to the user query; these chunks will be used as context in the prompt, in order to give the language model the external knowledge needed to correctly answer to the user;
3. Generation: a prompt is crafted with the user query, the extracted content and the necessary instructions necessary for the model to understand the task and give the output in the desired format; so, the prompt is given as input to the LLM, which generates the response to the user query; in case of ongoing dialogues, any existing conversational history can also be integrated in the prompt, creating a model that can handle a multi-turn dialogue, based on external knowledge.

Considering the application of this thesis - a question-answer chatbot for customer support, which helps users answer questions about the ERP system and the company's proprietary documents - a standalone pretrained LLM is not able to answer the users' questions, given that it wasn't trained on this data. In this situation, the LLM response can only state that it is not able to answer to the question due to lack of information, and even worse, the model could answer hallucinating, giving only incorrect and misleading information. The RAG technique addresses this problem, retrieving relevant information from the company's documents based on the user's query, then providing this information as additional context to the LLM input, to generate an accurate response based on specifics found in the relevant documents. This method solves some of the main problems of LLMs when applied to specific domains: the lack of knowledge and the lack of up-to-date information. In fact, LLMs are not trained on proprietary data, but only on publicly available data, so they cannot accurately answer questions about a company's internal documentation. They also do not have access to information about events occurred after they were trained. The only solutions in order to solve this issue is to fine-tune the model, training

it on the proprietary data or implement a RAG system. Given that the fine-tuning is more complicated and costly, and given the achievements of RAG systems, this second technique is generally preferred as the starting point for an application in the current market. In fact, RAG applications can supply the LLM with timely information from an updated data source of the proprietary data, allowing the model to answer to questions about the specific domain, receiving always the most updated answer [16].

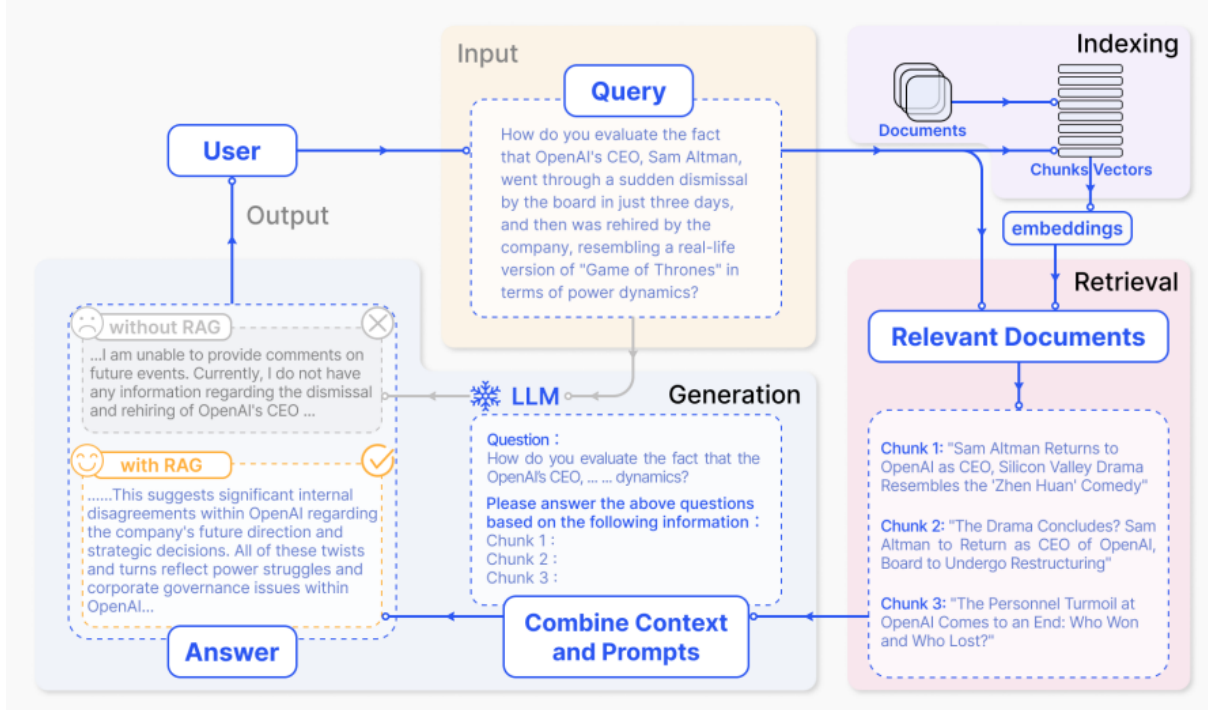


Figure 2.1: **Naive RAG [15]**: example of a RAG process from the 'RAG survey' paper, showing the main steps of this technique. The user gives as input a query, which the system is not able to answer to without external knowledge, thus, the RAG process becomes fundamental. The external knowledge, made by documents, is spitted in chunks encoded into vectors and the chunk vectors are indexed and stored in a vector database. When the user query arrives to the system, the RAG embeds the query with the same embedding of the chunks and extracts the most similar ones based on semantic similarity. Then, the initial question and the chunks are combined together obtaining a prompt that is fed to the LLM to generate the final answer. Thanks to the RAG process, the model now knows the answer and can give a response to the user.

The external knowledge used by the RAG system can be structured or unstructured and the source of this information can be databases, documents, websites or any other fount of data, which is first processed using NLP. In this application, the knowledge will be given in the form of documents, that contain the company's private information on how

the system works and how the user can act upon it. In order for the retriever to not only get the external knowledge, but also obtain the most up-to-date information, it will be necessary to set up an automated workflow that periodically ingests the updated content. Additionally, some metadata can also be used to tag the outdated information, allowing the system to prioritize during retrieval the more recent and relevant documents. After a domain analysis, it was considered valuable to incorporate new information as soon as it became available, and in the system, this would consist in implementing, once a week, small updates to the knowledge base. This external and up-to-date data integration in the system and in the LLM prompt also reduces hallucinations, making the system more reliable and robust. Another important aspect of a RAG system is its flexible architecture, which allows the RAG to be more easily integrated with other technologies such as prompt engineering, fine-tuning, and reinforcement learning, for better retrieval and generation results.

Regardless of the advantages of this system, the Naive RAG still encounters many problems in the three areas of the process: there can be retrieval challenges, extracting irrelevant information and misleading the model answer; also, the model can still be affected by hallucinations and its generations depend immensely on the selected information from the database, which can be not only misleading, but also repetitive and confusing, leading to responses that can be incorrect, repetitive, or of low relevance for the application [15]. In some cases, the retrieved information could also conflict with the model's internal knowledge, creating confusing and inaccurate answers. Thus, even if the basic RAG explained so far partially solves some of the LLMs problems such as lack of information and hallucinations, it is not a flawless technology and it is possible to have many different improvements, which will be explained in the following chapter, unraveling different possibilities to implement advanced RAG systems.

In this chapter were introduced the key concepts of NLP, machine learning, large language models, and RAG systems, providing the foundation for understanding the thesis work. Regarding the RAG, which is the focus of the implemented solution, some advanced techniques and more detailed concepts will be explained in the following chapter, referring to some state-of-the-art approaches. The following chapters will delve deeper in RAG techniques and the implementation of a RAG-based chatbot within an ERP system, in order to enhance customer support and reduce support ticket volumes and human resources.



## 3 | Retrieval-Augmented Generation (RAG) Advanced Techniques

In the basic Naive RAG, as seen before, there are three simple main steps: the indexing, the retrieval, and the final generation. Each of these steps can be improved, enhancing the overall system. Let's consider each of these three steps.

### 3.1. Advanced indexing

An advanced RAG system refines its indexing techniques, improving the pre-processing step that directly impacts indexing, and using optimization methods in order to improve the retrieval process. To improve indexing it is first necessary to enhance the data by preprocessing and segmenting it into manageable and meaningful units, so this step of the RAG makes it possible to act upon:

1. **Preprocessing and augmentation of the original data:** ensuring that the original data is cleaned and normalized, to improve its quality and usability, as well as augmenting the dataset by enriching it with meaningful additional information such as paraphrasing sentences or integrating additional image descriptions; curating the dataset to have high-quality knowledge is the first and most crucial step for the system improvement, given the data quality principle of *"Garbage in, Garbage out"*, that states the huge influence of the dataset and its preprocessing on the final results;
2. **Chunking strategy and chunk size:** choosing the chunking strategy which ensures that the chunks are meaningful and contextually coherent, adjusting the chunking based on the task and on the LLM's context window, retrieving then the chunks instead of the entire documents, having in the prompt, more targeted context from which to generate responses; this allows not only to have more precise information, but also to be able to handle long documents, that may exceed the

LLM's input limit. At this step, additionally, based on the chosen chunking strategy, it can be really important to tune the chunking size to a dimension that retains as much relevant information as possible, while losing some less relevant data, in order to capture only the essential and necessary context;

3. **Improving the quality of the embeddings for dense vector indexing:** this step can consist in finding the best pretrained model for the task, in order to better embed the chunks and capture semantic meaning and relationships; it can also consist in fine-tuning embeddings on domain-specific data, getting even better results, especially if the embedding can be retrained periodically to reflect changes in the corpus or adapt to new data;
4. **Adding metadata:** providing additional context and structure to the indexed data, improving the retrieval accuracy and relevance; examples of metadata include date and time, title of the document, number of chunk in the document, summary of adjacent chunks, keywords, and other custom metadata, based on the task.

These optimizations can be considered part of the pre-retrieval process improvement, optimizing the indexing structure by enhancing the quality of the content being indexed.

## 3.2. Advanced retrieval

The second step is to enhance the retrieval process, that can be done by choosing the best retrieval technique for the task at hand and by performing hyper-parameter tuning to select the number of chunks to retrieve, in order to extract the most relevant information, balancing the correct amount of context to give to the model; additionally, the process can be further refined using some other techniques in the pre-retrieval and post-retrieval stages [15]:

1. **Pre-retrieval process:** query optimization consists in making the user's original question clearer and more suitable for the task, with common methods including query rewriting, query expansion, and other techniques; this step can include formatting the query, using another model to transform the question or extracting keywords to use for retrieval [18]. The goal of this technique is to handle ambiguous or incomplete queries in order to improve the retriever, for example, by using multi-query rewriting to generate different versions of the same question so that relevant information is more likely to be extracted despite the lack of clarity and context in the original user input;
2. **Post-retrieval process:** once relevant context is retrieved it is important to in-

tegrate it correctly with the query, and the main methods include the reranking and filtering of extracted chunks and the context compression in a curated prompt, combining the most relevant information in a concise and coherent format for the next stage of the process.

### 3.3. Advanced generation

In the final stage of the RAG, the generation process plays a crucial role on the overall quality of the system and two key choices can influence significantly the performance at this stage of the pipeline: the selection of the LLM used for the generation and the tuning of its parameters, such as temperature, which controls the randomness of the output, and the maximum input token limit, which establishes the amount of context the model can process at once. These choices are mandatory to set up a RAG system and they have to be made accurately, given that they directly influence the generation performance and relevance, but there are also other important techniques that can be added in an advanced RAG to improve this final step of the system:

1. **Prompt Engineering:** working iteratively on the prompt, integrating accurately the most relevant information in the instructions given to the model in order to specify the task, the knowledge necessary to use to answer correctly, and the desired output format of the response;
2. **Model Fine-tuning:** the LLM used for the generation step can be fine-tuned on domain specific data to improve its ability to generate contextually appropriate responses; using task-specific data, such as question-answering datasets, it is possible to better align the model to the desired behavior. In fact, while RAG enables the use of external knowledge giving great results in dynamic environments by offering real-time knowledge updates, the fine-tuning is more static, requiring retraining and high computational resources, but it facilitates the customization of the model's tone, behavior, and style, thus the fine-tuning can be complementary to the RAG implementation, enhancing the model capabilities at a different level [15];
3. **Feedback Loop Integration:** implementing a feedback mechanism where the generation results are evaluated, automatically or manually, and used to fine-tune the model or adjust it in order to get better quality outputs; some popular methods are: Corrective Retrieval-Augmented Generation (CRAG), which self-corrects the results of the retriever based on an evaluator in order to improve the extracted documents for the generation; and at a different level the Reinforcement Learning from Human Feedback (RLHF), a machine learning technique that allows to integrate

human preferences into the training process in order to align the model's responses with the desired outcomes.

By continuously evaluating the quality of the generated outputs with quantitative metrics or by gathering human evaluation, it is possible to use the feedback to guide further improvement, analyzing which are the weak parts of the system and how they can be improved. A common method used in production is to deploy two different versions of the model, the ones that, as a conclusion of the final analysis, work better, and use A/B testing, comparing the performances of the two versions, based on the satisfaction of the final users. Another way could be to use LLM as a judge for the different parts of the system, for instance, judging the quality of the extracted chunks or the quality of the final response, given some criteria; and it could also act as post-generation validation, comparing the output against the retrieved chunks, to ensure factual consistency and correct alignment.

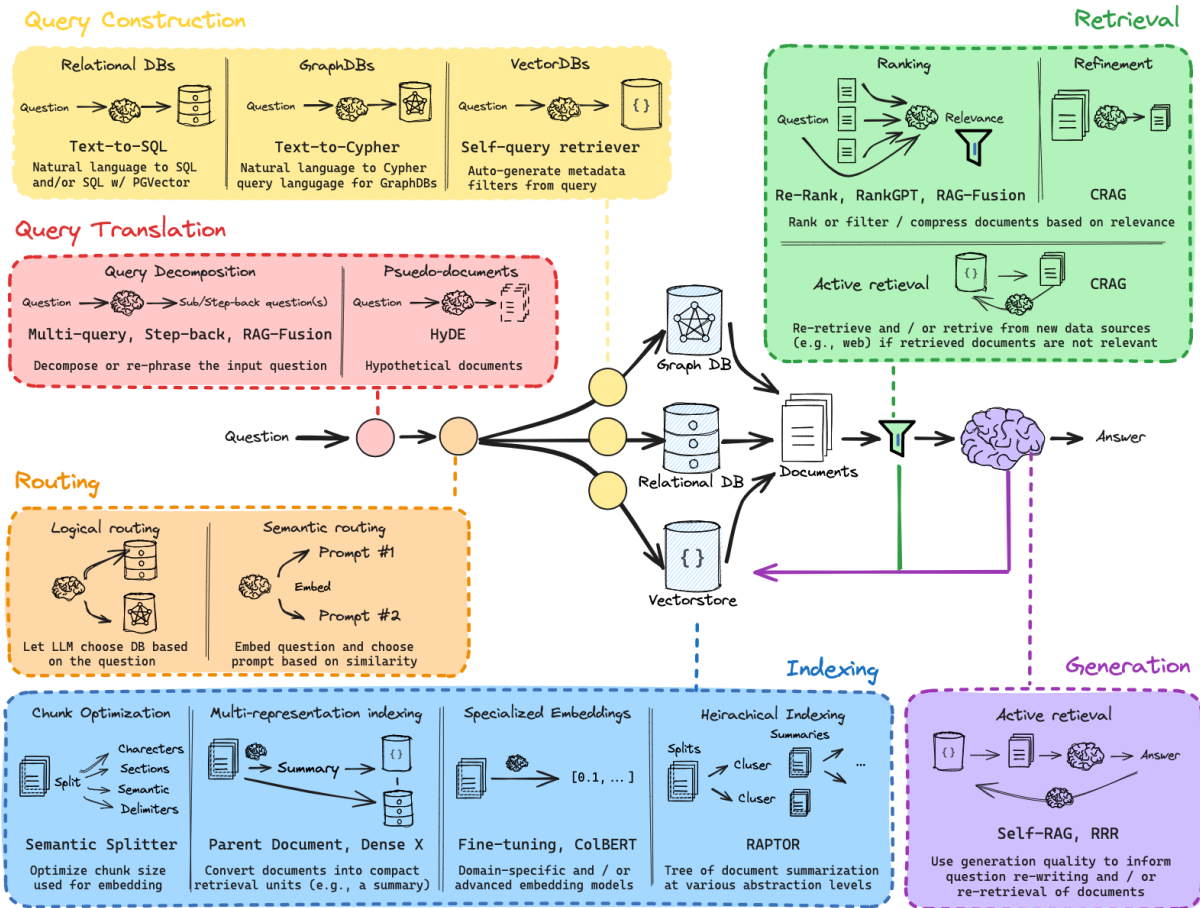


Figure 3.1: **RAG Techniques [17]**: basic and advanced RAG techniques that can be used to improve both the pre-retrieval and the post-retrieval process, enhancing the system as a whole, getting better performance and also improving the efficiency of the pipeline.

As it can be seen in the image, there are many other techniques, not mentioned here, that can be set up in order to get a better RAG system. One of the other aspects that can be improved is the routing of the RAG system, enhancing how the RAG navigates through different data sources, selecting the optimal path for a query, but given that in the task at hand, only one data source is used, the analysis will disregard this kind of improvement. Active retrieval in the generation step, which can be seen in the figure above, will also be disregarded, not being used in the proposed methodology, but will be shortly discussed as one of the future developments that can be considered to further improve the system.

Some of these implementations are necessary to get better results in the task at hand, given that the Naive RAG implementation, even if very useful, has some limitations. Without any of these advanced techniques, the implementation still suffered intensively from hallucinations and struggled to retrieve the correct information needed to answer the query. Some experiments during the implementation, which will be explained in the following chapter, showed the fundamental role of some advanced techniques in order to tailor both retrieval and generation processes to the specific domain of the task and obtain an overall better RAG system with higher performance. After choosing the best RAG pipeline implemented so far for the customer support task to make the system a real chatbot, it was also necessary to implement a memory, in order to have accurate multi-turn conversations, and also an interface in order for the system to be actually used by the final user.



## 4 | Proposed Methodology and Implementation

### 4.1. Requirements and implementation strategy

Prior to analyzing the implementation choices and the obtained results, let's first delve deeper into the requirements of the application and the final goal of this thesis, starting from a use case example, to have a clearer understanding of the task, to keep in mind during the following explanations.

#### 4.1.1. Requirements analysis and extended use case

In modern software applications, user support systems often include chatbots to assist users in resolving issues and these chatbots are designed to not only offer solutions based on documentation, but also to handle more complex tasks, such as creating support tickets. The following example showcases a chatbot-driven system for issue resolution and ticket creation, highlighting its requirements and functionality.

---

#### Chatbot Extended Conversation Example:

**User:** I'm unable to create a new document of type TD24 when I open the invoice creation screen because the "Submit" button doesn't work. Can you help me?

*Retrieval-Augmented Generation system pipeline activated to answer to the user.*

**Chatbot:** - *explanation from the documentation* – Does this resolve your issue? If you need more assistance, do not hesitate to ask me more about it or anything else you require.

**User:** - *continue a multi-turn conversation* -

**Chatbot:** - *after a certain number of interactions, if the user still tries to solve the problem by interacting with the chatbot* - Do you want to create a ticket for this issue?

**User:** Yes.

*The ticket creation process is triggered using a dialogue task-oriented chatbot to fill the slots necessary to create as clear as possible a ticket to send to the help desk department of the software.*

**Chatbot:** Alright, let's create a ticket under the name "**User\_Name**" with the following message: "Hello, I've encountered an issue with the creation of a TD24 document where the 'Submit' button is not functioning." Do you confirm this?

**User:** Yes.

*Continue the conversation, asking all the necessary information to write the ticket.*

**Chatbot:** Now, here's a preview of your ticket, including the category and detailed description, gathered from our previous conversation. Please confirm if everything looks correct.

*User confirmation.*

**Chatbot:** Great, you will now be redirected to the ticket creation page, where you can see a preview of your ticket. On this page, you will have the opportunity to review all the details, including the categorization, description, and any attachments. If needed, you can modify the information before finalizing and submitting the ticket to the help desk. I hope I was able to assist you.

---

This use case demonstrates how a chatbot can assist users in troubleshooting application issues and, if needed, guide them through the process of creating a support ticket. The chatbot initially offers potential solutions based on documentation, and if the problem persists, it helps the user create a fully detailed ticket by asking for relevant information (e.g., user details, description of the issue, screenshots). The ticket is then pre-filled and ready for submission, streamlining the support process.

Based on the chatbot conversation example, created when designing the system and analyzing the requirements, the system must achieve the following goals:

#### 1. Issue Identification and Information Retrieval:

- The chatbot must be capable of understanding user queries, such as application-specific issues, through natural language processing (NLP);
- The system must retrieve the most relevant information from the documentation in order to then use it to address the user's problem.

#### 2. Multi-Turn Conversations:

- The chatbot should support multi-turn conversations, maintaining context throughout the dialogue to refine or elaborate on user queries;
- The system should have a memory in order to keep track of conversation history, and use it in the model prompt, combined with the retrieved documents and the crafted instructions to accomplish the task, solving the user's request and reducing as much as possible the ticket creation, with a goal of at least 20% reduction of the volume of support tickets.

### 3. Ticket Creation Workflow:

- The chatbot must also have a dialogue task-oriented part to guide the user through a ticket creation process, asking for necessary details in order to fill up the slots for the ticketing;
- The chatbot should accurately pre-fill the ticket form with the collected information and offer a preview for user confirmation before submission, reducing human error as much as possible in this process.

### 4. User-Friendly Interface:

- The chatbot interface must be user-friendly, minimal, and responsive, while allowing a multi-turn, human-like conversation. It could also allow the user to leave feedback on the conversation or on different turns of the conversation in order to have human feedback for future system evaluation and improvement;
- The chatbot has to clearly communicate the progress of the conversation, such as triggering the ticket creation process or presenting a final ticket preview; and the user should have the ability to modify the ticket before submission.

The main goals are, first, the creation of the chatbot capable of accurately answering to the user questions using private data, and second, the ability of the system to also guide the user through the ticket creation.

In the subsequent sections of this thesis, the focus will be on the primary requirement of **issue identification and information retrieval**, utilizing the available data to accurately respond to user queries. This will involve exploring methodologies for effective information retrieval and answer generation, enhancing the LLM with a RAG system. The requirements of **multi-turn conversations** and the **user-friendly interface** will also be briefly addressed towards the end of this thesis, describing the choices made to implement a memory for the chatbot, leveraging advanced prompt engineering techniques to maintain the conversational context; and outlining the creation of a simple minimal interface to allow users to interact with the system. The created interface will be further improved by future developments, for example, by integrating a feedback system to leverage human feedback and gather insights from real conversations to refine the system.

While this thesis prioritizes these three main requirements, future work will need to expand on the **ticket creation workflow** requirement, further extending the system described here. By also developing this aspect, it will ensure the system fully achieves all the goals identified during the requirements analysis of the system.

### 4.1.2. Implementation strategy

The implementation of the proposed solution is guided by two fundamental principles: a data-driven approach and an iterative improvement.

First, the solution has to consider the available data, making data informed decisions in order to implement a system that is tailored not only to the specific task at hand, but also to the characteristics of the domain and the associated documentation. Following this principal, the implementation tries to maximize the system's performance adapting to the type of data used in the context, so the solution cannot be an absolute best solution, but only the best one found in the specific context and for the available dataset. If the documentation and the domain drastically changes in time, new analysis and improvements will be needed to have a system at least as good as the one proposed here.

Second, the implementation process follows an iterative strategy. By starting with a minimal viable solution and incrementally enhancing it, the approach maintains a balance between performance and cost. Budget constraints are consistently taken into account, with a focus on exploring various possibilities and achieving meaningful progress while minimizing the costs. This principle guides in fact the choice of using a RAG system, with a simple naive implementation, starting with basic techniques and an off-the shelf model, and moving towards more complex solutions only when necessary and if the trade-off is acceptable. The iterative improvements that refine the retrieval process and the generation will be guided by human evaluation, and also by some quantitative metrics, that for this specific application revealed most aligned with the human considerations, and the goal will be to achieve a sufficient chatbot quality while minimizing cost.

This overall implementation strategy ensures that the system evolves in a controlled and efficient manner, delivering measurable improvements without exceeding resource limitations. Thus, the design space is explored systematically, starting with the simplest and least expensive solutions and progressing toward more complex and costly ones, as necessary. The exploration of the solution starts from the RAG implementation, then enriched mainly with Retrieval Information techniques and Prompt Engineering guided by the augmented and improved dataset, trying to deliver a robust solution tailored to the specific requirements previously explained.

When applying again the same strategy in the future for more improvements, or even when applying it to other tasks and projects, it is also important to know when to quit. In fact, knowing when to quit is as important as understanding in which order to pursue the strategy to explore the design space at minimum cost possible, and it is important to consider that each model has limited potential, so it cannot also behave perfectly on

a task. If the results are not satisfactory enough and the model or the system itself doesn't improve, it is necessary to quit, change the model or the entire pipeline, and analyze the problem from the start, searching for a different methodology. In the proposed implementation it is important to notice that the LLM is almost interchangeable, so the model can be changed at need, using public or private models, considering costs and the model actual improvement on the task, especially after trying to fine-tune it on the specific domain.

The proposed methodology uses off the shelf models combined with RAG and Prompting, but in future developments, following this iterative strategy, the system can be enhanced, first by fine-tuning the model on the specific data of the application, and then by performing continuous pre-training, training a lot more the model if budget constraints allow it. Evaluating the cost-quality trade-off and the time to success of the model, it can be decided when to stop and when a model improvement is actually worth the cost, or when a model change could be preferable. It is important to *climb the ladder in order*, because maybe it is not necessary to spend a lot to get the desired quality system.

## 4.2. Data creation and preprocessing

Regardless of the application, it is necessary to spend some time and effort to investigate the source data and pre-process it to convert it into a form in which we can use it for the application. The data source for this customer support chatbot is made of technical documentation about the software, which is available in PDF format. Thus, the first important step was to handle the complex PDF structure and clean the data appropriately to extract a plain text in Italian, that can be then tokenized and used for the application.

### 4.2.1. Corpus composition

Given that the proposed implementation relies on the Retrieval Augmented Generation (RAG) technology, and considering, as explained before, the crucial impact of information retrieval on this kind of applications, it is necessary to first prepare the right data corpus in order to then retrieve the information required to answer the user queries. The data is entirely dependent on the application, and for a customer support chatbot in an ERP system, the corpus composition may include private documents, product manuals and specifications, troubleshooting guides, and frequently asked questions (FAQs).

From an analysis conducted together with domain experts, it was possible to identify the relevant content for the chatbot, gathering also insights on the types of questions the users

are likely to submit. The identified data source is made up of almost 300 private documents, including software manuals and specifications, and some troubleshooting guides for the most common problem resolutions. Due to the complexity of the ERP system and the diversity of questions, which vary depending on the range of functionalities across various business domains, on the software version and customization, and on specific customer requirements, it was not feasible to create a set of FAQs. However, over time and with the help of domain experts, such information can be gathered and curated to identify significant questions and their resolutions. Given the decisive role of the data corpus in this type of application, it becomes a priority to further improve and integrate the dataset with high-quality information, in order to improve the system.

#### 4.2.2. Augmented Dataset Creation

Once identified the data corpus, the next step is to extract the required information from the raw data, parsing the PDFs to transform the unstructured data into a format that can be retrieved and effectively used by the RAG system. At this step, it was decided not only to get the data from the PDFs in the desired format, but also to augment the information, more precisely by preprocessing the extracted images with an LLM, generating the description of software screen images and embedding these descriptions within the sections of text, exactly where the images were originally present.

This augmentation was considered fundamental after the consultation with the domain experts, since the manuals and the troubleshooting guides often rely heavily on the images to convey the information, which is not always described in the textual data. To have all the information necessary for the chatbot to answer to the questions, it becomes crucial to also get the screens textual description, integrated in the correct position in the text, to enhance the clarity and completeness of some explanations related to specific actions and processes.

Thus, at this step, the raw documents are processed to extract both the textual content and the embedded images. The off-the-shelf `unstructured` library is employed to extract structured content from the PDFs, breaking down the documents into elements such as `Title`, `NarrativeText`, `Table`, `Image`, with also the ability to customize the parsing strategy, customizing it based on the specific requirements. The `partition_pdf` function was used to segment PDFs using a document image analysis model, specifying the Italian language for the optical character recognition (OCR), selecting the parsing strategy and enabling the `extract_images_in_pdf` parameter. The function identifies different elements in the PDF and saves the extracted elements in JSON format.

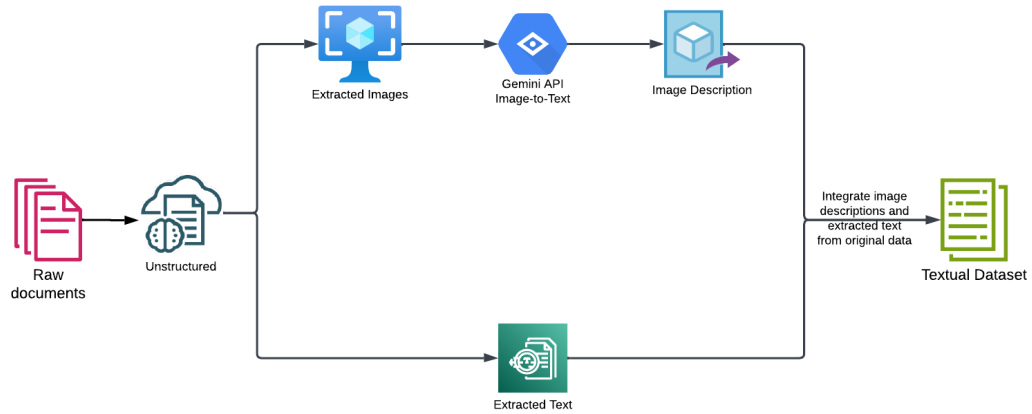


**Figure 4.1: Process for creating an augmented textual dataset:** *Raw documents are processed to extract both textual content and embedded images, which are converted into descriptive text using an image-to-text AI model and then integrated to form a comprehensive textual dataset, enhancing the original document's information*

This output was then processed following two different paths to get to the augmented textual dataset for the RAG system:

1. **Selecting the relevant textual content:** elements under the categories of `Header`, `Footer` and `PageNumber` were removed, considering them irrelevant, while the categories `NarrativeText`, `Title`, `Image`, `ListItem`, `Table`, `UncategorizedText`, were kept as textual content, and processed in the next step in order to make them more suitable for the application, removing some unwanted patterns that appear especially in the `UncategorizedText` and in `Table`, such as instances of tables that contain PDF indices;
2. **Augmenting the textual content with image descriptions using LLM:** the images extracted by the `unstructured` library under the structure type `Image`, as any other element has a lot of associated metadata, such as `id`, `page number`, `PDF file name`, `last modification date`, `language`, and other details; the most important elements for this process are the element `id` and the image path. The element `ID` served as a placeholder during extraction, while the image path linked the image file to its `JSON` element. This facilitated the identification and reintegration of textual descriptions into the correct positions in the text flow. The extracted images are saved in `.jpg` format and processed using the `Pillow` library, which loads the image from file, then passed together with a prompt to the LLM to obtain the image description. After experimenting with various models, such as `gpt-4-vision-preview`, `gemini-1.5-pro-latest`, and a pipeline combining `llava` (for English descrip-

tions) with `mbart-large-50-many-to-many-mmt` (for Italian translations), it was determined that the open-source combination did not produce satisfactory results, introducing too much noise in the process. Comparing, then, the quality and costs of GPT and Gemini models, the final decision was to use the `gemini-1.5-pro-latest` model for all the documents image processing. During this phase, different prompts were tested, analyzing the results manually; a simple prompt was identified to work better on the task, passing to the model the image and the simple instruction to "describe the main parts of the window screen from the provided image". The prompt ensured good descriptions of the relevant parts of the software screens, then integrated with the rest of the textual content.

The figures **Figure 4.2** and **Figure 4.3** illustrate this process of data augmentation with an example use case. The first figure, **Figure 4.2**, shows the workflow for generating the augmented dataset, including the chosen prompt, the simplest one, provided to the Gemini model together with the image, to obtain the image description. Analyzing this augmentation process, it is clear that the Gemini description successfully describes the main parts of the window from the application, extracting correctly the exact names of the flags, their functionalities, and how the user can act upon them. There is also some noise in the result, for example in this case, the last part of the description doesn't correspond to the image, so the model is hallucinating, but in the final result it is not so relevant, being a part that will never be asked by the user, so it doesn't influence the quality of the system responses. The second figure demonstrates in fact the improvements achieved with the data augmentation process.

Analyzing both responses in **Figure 4.3**, the first and most notable improvement is the fact that the new response explicitly mentions the option as `Previsione in % satur.` and the term directly matches the label seen in the interface within the uploaded image (differently from the previous answer which specifies more generically the flag, which could sometimes make unclear to the user which specific flag in the software is actually being addressed). In this specific case, the original answer also lacks important information, focusing only on a part of the user question, potentially not associating correctly some parts of the document chunks obtained with the splitting, while in the improved version, the response aligns more with what the user needs to understand and accomplish, focusing on the relevant checkbox option and clarifying its function. Thus, integrating data augmentation into the process, the response becomes more precise, leveraging the exact terminology of the software, minimizing any potential misunderstanding, and ensuring more efficient and clear responses, guiding the user to understand the setting and complete the task.

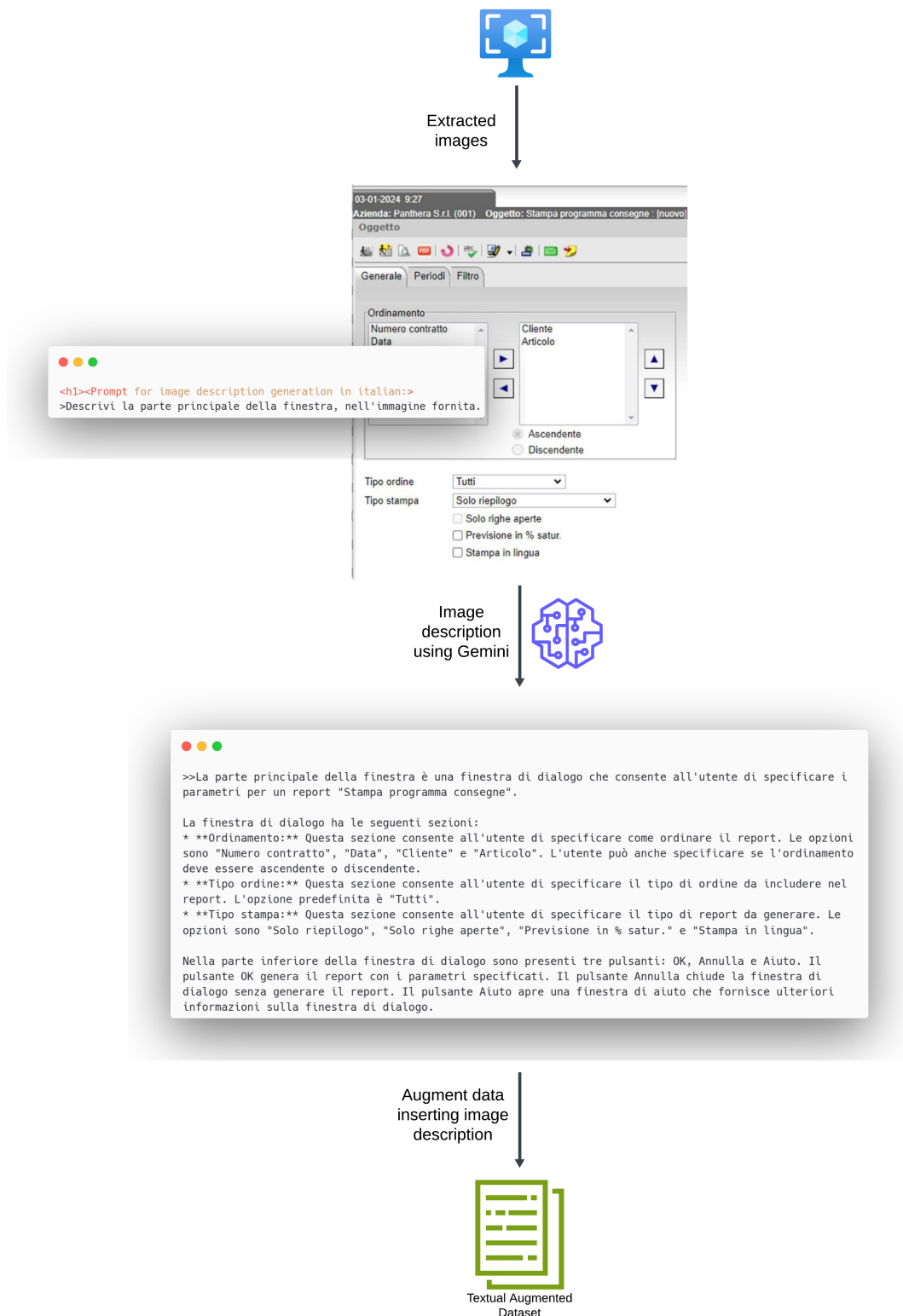


Figure 4.2: **Data augmentation process:** *Passing to Gemini the extracted image and the simplest prompt, the result is a good description of the main parts of the interface screen, adding to the textual data precise terms and labels from the software.*

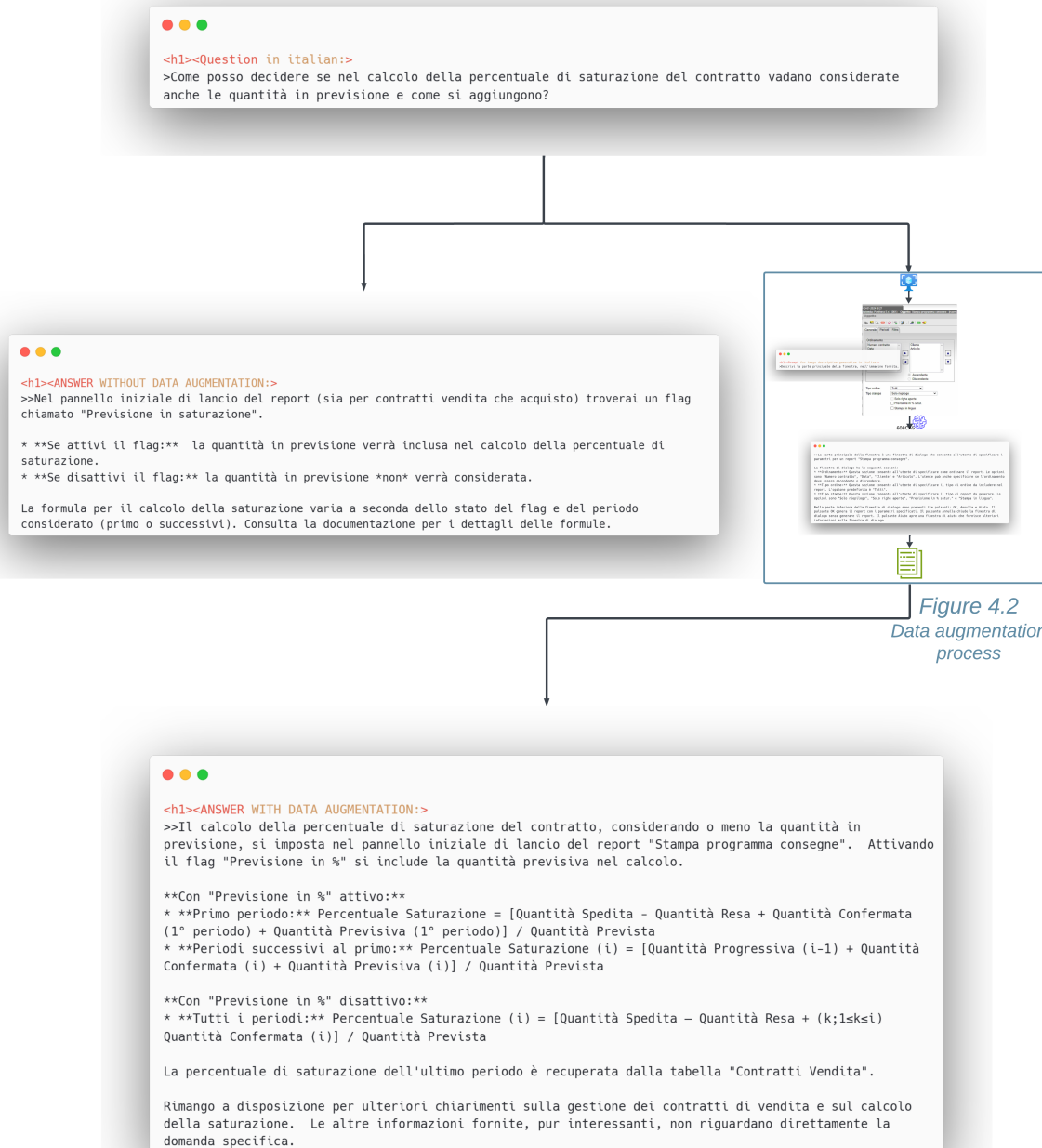


**Figure 4.3: Answer comparison with and without data augmentation:** Following the same use case of the previous figure, there is a noticeable improvement of the obtained result, due to the introduction of the image description in the textual content of the documents, having an exact terminology match between the answer and the software interface, and obtaining a more precise response, having an improved alignment with user needs, guiding the user to the understanding of the functionality of the mentioned flag, and ensuring that the user can activate it, specifying the precise checkbox.

After the augmented textual dataset creation, a quality control is performed, checking the quality of extracted text and performing a cleanup and normalization of the data, preparing it to be effectively used in the application. The following subsection delves deeper into the data preprocessing experiments undertaken to improve the quality of the textual content and discusses the results, evaluating the impact of the preprocessing pipeline on the chatbot’s final responses.

### 4.2.3. Data preprocessing experiments

The best practices in the NLP field indicate the importance of performing at least a basic data preprocessing of the plain text extracted from the PDFs, given that the parsing of PDFs documents may introduce inconsistencies and unwanted patterns in the extracted textual content. The main operations to perform in order to cleanup and normalize the text are the following:

1. Remove undesired patterns, such as page numbers, repeated text, indices, headers and footers;
2. Remove duplicate spaces and newline characters, standardizing white-spaces;
3. Fix common errors through regex rules or spellcheckers;
4. Remove punctuation and special characters.

Taking into account these best practices and analyzing the extracted augmented plain text, different pre-processing pipelines were designed to assess the importance of different actions applied to the data, evaluating the results with some automatic metrics and human feedback within an end-to-end system, employing the simple RAG approach.

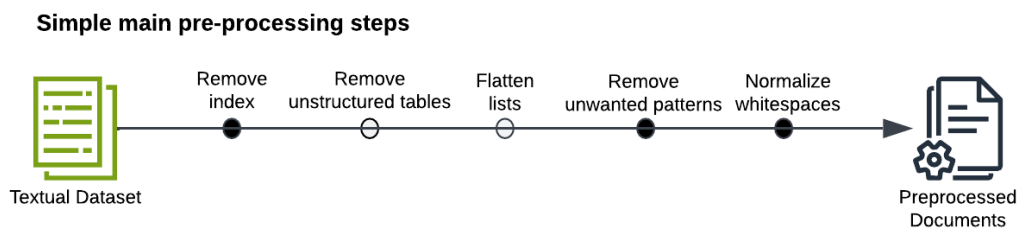


**Figure 4.4: Textual dataset main preprocessing steps:** *a simple, but partial preprocessing of the dataset, performing only the highlighted steps, removing the index, unwanted patterns, and normalizing the white spaces.*

An initial experiment is conducted using a simple partial preprocessing pipeline focusing only on the most evident elements to clean up from the extracted data, in particular, the PDFs indices, unwanted patterns such as some residual footers and headers, specific and recurrent date formats, and special characters like extra dots. Additionally, this first experiment included whitespace normalization, removing duplicate spaces and excessive new line characters. These operations were considered the most important ones after human evaluation of the extracted augmented dataset and after some preliminary experiments, but the final evaluation of the simple RAG pipeline with this partial simple preprocessing didn't hold better results than the pipeline without any preprocessing at all, posing some questions on the real impact of preprocessing on new NLP applications, that use robust algorithms and LLMs.

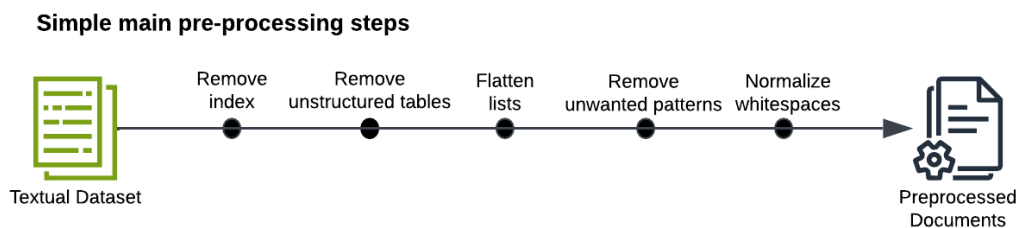


**Figure 4.5: Textual dataset simple preprocessing pipeline:** *differently from the partial simple preprocessing pipeline, all the preprocessing steps showed in the figure are now executed, adding in fact the removal of unstructured tables and flattening the lists, normalizing even more the textual content.*

The inclusion of some additional preprocessing steps, such as removing unstructured tables and flattening the lists, did not lead to significant changes in the results, but showed some little improvement. This enhancement was observed not only in the evaluation metrics derived from the end-to-end system tested on a synthetically generated evaluation set, but also through human analysis of the responses. The refined preprocessing pipeline contributed to slightly more accurate and coherent outputs, demonstrating a modest, yet valuable improvement in both automated assessments and human evaluations of the chatbot's performance. These enhancements likely contributed to better data quality, making the textual dataset more suitable for retrieval and generation tasks, and regardless of the small improvement, these additional steps did not introduced significant complexity to the preprocessing and were quick and straightforward to implement, ensuring a cost-effective enhancement. Thus, these changes were adopted for the final preprocessing pipeline, balancing system performance improvements with practical considerations.

Another experiment was to execute all the steps of the simple preprocessing pipeline, except for the whitespace normalization, in order to not remove additional white spaces that could represent the original PDFs structure. Without removing the whitespace, the results show slight improvements when using recursive character splitting chunking, but these conclusions are directly applicable only to this type of chunking, and cannot be generalized also to the semantic splitting approach. This improvement can be attributed to the nature of recursive character splitting, which relies on predefined separators, which in order are: `\n\n`, `\n`, `" "`, `" "`. By retaining the white spaces `\n`, the textual structure remains more intact, enabling the splitting process to better leverage the original text layout and paragraph structure. Thus, this preservation of whitespace contributes to a better alignment of chunks with their contextual meaning, enhancing the overall system performance in this specific case. This is a relevant improvement to take in consideration for pipelines opting for recursive character text splitting over semantic chunking. These strategies and the possible implications will be further discussed in the following chapter about chunking strategies.

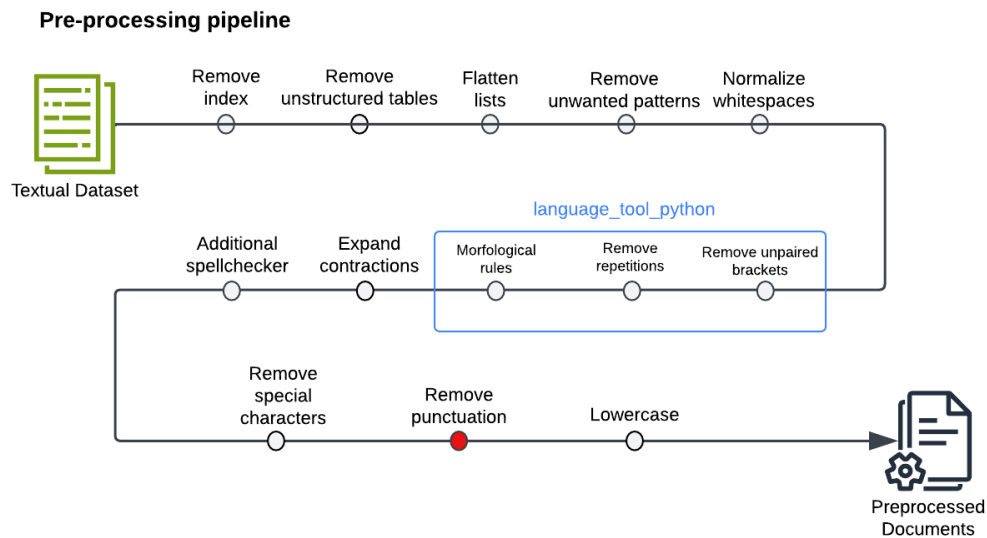


**Figure 4.6: Advanced textual dataset pre-processing pipeline:** *using a much more complex preprocessing pipeline, the previous result is further processed, trying to remove additional unwanted patterns, correct misspellings, eliminate punctuation, and convert all text to lowercase. In general, it is important to note that the punctuation removal may be optional depending on the chunking strategy. For instance, if sentence splitting is part of the chunking process, retaining punctuation could be essential. Therefore, this step is highlighted, because it should be carefully considered based on the subsequent stages of the application pipeline.*

The implemented advanced preprocessing pipeline includes a variety of techniques, using some Python libraries and implementing some custom algorithms tailored to the dataset, given the limited availability of effective methods for processing Italian text. Libraries such as `language_tool_python` and `spellchecker` were used to normalize the text and correct it grammatically and syntactically, removing misspelling errors and other issues identified by the tool and considered as such by a thorough human analysis. Morphological aspects and misspellings were verified more than once using the two mentioned libraries, as these tools often demonstrated inconsistencies in handling Italian text effectively, so their abilities were supplemented wherever possible. The contractions, on the other hand, posed a unique challenge due to the lack of suitable libraries for Italian, so a custom function was developed, by first finding all the words that contain an apostrophe in the available documents, and then passing them through an LLM, specifically `gpt-4o-mini`, which for each word given as input returns in a specific format the contracted word together with the respective extension in Italian. The formatted output is then further processed in order to systematically substitute each contraction in the documents with its expanded version. These advanced preprocessing operations were then integrated with some additional steps, such as removing special characters, converting text to lowercase, and when possible, based on subsequent pipeline stages, eliminating punctuation. All these techniques lead to a more sophisticated and complex data preprocessing, but the results, as detailed in Table 4.1, show no significant benefits over a simpler pipeline.

The experiments conducted on different variants of the advanced pipeline, applied as a whole or only partially, revealed that these steps influenced only marginally the chatbot responses, and there weren't noticeable improvements. On the contrary, the best results were achieved using the simple preprocessing pipeline. Moreover, it is important to consider the fact that the advanced pipeline was more resource-intensive and time-consuming, further justifying a preference for the simpler implementation and confirming the diminishing importance of extensive preprocessing in modern NLP tasks. The obtained results align with the general trend in NLP applications, where pre-processing pipelines have become shorter over time, making extensive pre-processing less essential for application performance. The shortening of the data preprocessing pipeline in NLP and generative AI applications can be attributed to several factors:

1. **Automation and advanced libraries:** modern data science libraries have introduced preprocessing tools that automate many tasks. The `unstructured` open-source library offers the possibility to design ingestion and pre-processing of diverse data formats, including images and text-based documents such as PDFs, optimizing data workflows for LLMs, shortening the data preprocessing pipeline [9];

2. **Improved algorithms and pre-trained LLMs:** algorithms have become more robust to raw data, for instance, in generative AI models, like Llama, Gemini and GPT, the models are designed to handle a variety of input formats without extensive preprocessing, reducing the necessity of traditional preprocessing steps; in fact, these models are pre-trained on diverse datasets and are robust, working effectively even with raw or lightly processed inputs;
3. **High-quality data focus over extensive preprocessing:** modern NLP and Generative AI applications prioritize the inclusion of high-quality and relevant data in the creation of the data corpus, rather than applying a multitude of preprocessing steps, given that having high information quality of the source is more effective than complex data preprocessing.

In the following table were gathered all the results of the various experiments, with some observations of the impact of various data preprocessing strategies and pipelines on the performance of a basic RAG pipeline with Gemini. The RAG pipeline used to evaluate the preprocessing steps doesn't use any advanced RAG technique, it just receives the questions of the synthetically created evaluation set and uses a simple dense retriever with  $k = 4$ , to extract exactly 4 chunks from the FAISS vectorstore, the ones that are considered the most similar to the user question. Both the user question and the preprocessed textual content of the documents are embedded using the BAAI/bge-m3 embedding model. The documents are processed using recursive character text splitting, with chunk size of 3000 and chunk overlap of 400, before being embedded and saved in the vectorstore. Once extracted the most relevant chunks, on the generation part of the RAG pipeline is used a Gemini model, the `gemini-1.5-pro-latest` model and using a relevant zero-shot prompt, obtained after some prompt engineering, the model receives the user question and the extracted context and generates the final output. The evaluation set of question and answers contains approximately 100 pairs and it is produced synthetically with the testing framework, that will be explained at the end of this thesis in detail together with the choice of the evaluation metrics better aligned with human feedback for this application, in order to better understand the following provided results. While all the elements of the pipeline will be detailed in subsequent chapters, their mention here was necessary to clarify the evaluation setup and ensure a clear understanding of how the results in Table 4.1 were achieved. The following results start from a baseline obtained using the specified RAG pipeline with no preprocessing, subsequently compared with different levels of preprocessing:

| Preprocessing          | ROUGE-1 | BERT F1 | SemScore | Observations                                                                                                                                                                                                                                                                     |
|------------------------|---------|---------|----------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| No Preprocessing       | 0.375   | 0.726   | 0.899    | Initial baseline configuration with no preprocessing, using the metrics selected as the most aligned with human evaluation.                                                                                                                                                      |
| Simple Partial         | 0.374   | 0.725   | 0.897    | Partial preprocessing slightly reduced performance, even if from human analysis these operations were considered the most relevant to obtain a clear dataset. Poses some questions on the real impact of preprocessing.                                                          |
| Simple Pipeline        | 0.382   | 0.728   | 0.898    | Best performance is obtained with simple preprocessing, highlighting the trend of reduced preprocessing pipelines in new NLP applications.                                                                                                                                       |
| Simple with whitespace | 0.380   | 0.730   | 0.903    | Best performance specifically for recursive character splitting technique, keeping the simple preprocessing pipeline, but without whitespace normalization. Achieved maximum values on almost all metrics, enabling the specific chunking to better leverage the text structure. |
| Partial Advanced       | 0.379   | 0.727   | 0.898    | Stopped preprocessing at contraction expansion. The results showed minor improvement compared to advanced preprocessing but did not surpass simple preprocessing, probably introducing some noise in the process.                                                                |
| Partial more Advanced  | 0.381   | 0.727   | 0.898    | Supplement the first morphological check with an additional spellchecker, with the idea to reduce the introduced noise, and improve grammatical correction. Performance approached the simple preprocessing results, but with much more computational complexity and cost.       |
| Advanced Pipeline      | 0.365   | 0.723   | 0.895    | Advanced preprocessing reduced performance, adding punctuation removal and lower-casing. While normalization is important for reducing inconsistencies, it seems that over-normalizing may homogenize too much the text and reduce performance, depending also on the chunking.  |

Table 4.1: Summary of Preprocessing Pipeline Experiments and Results

From the results, it can be observed that the preprocessing has a limited effect on the performance metrics, such as ROUGE-1, BERT F1, and SemScore; and the best results are obtained across all metrics with the simple preprocessing pipeline. Advanced preprocessing was found to be both computationally expensive and less effective, and sometimes to even interfere with the chunking strategy. These findings demonstrate that focusing on lightweight preprocessing steps, while avoiding over-engineering, can be more effective in modern NLP tasks involving LLMs. The robustness of contemporary LLMs, with advanced pre-training on a variety of data, reduces their dependency on heavily preprocessed input data. Consequently, given these results, resources might be better allocated to other components of the pipeline, such as the steps of retrieval and generation.

#### 4.2.4. Chunking strategies

Chunking is the process of splitting large documents or datasets into smaller, manageable segments, called chunks. In the RAG system, the pre-processed documents are split using a chunking strategy, after which they are embedded and stored in a vector database. Instead of embedding the entire document, we split it into smaller parts of information, in order to enable the retrieval model to extract the relevant information more efficiently. Each vector in the database encodes the meaning of a chunk, allowing the retrieval system to not just match the common keywords between documents and user question, but also to retrieve the information based on the context [10].

In text, retrieval granularity can range from fine to coarse, including token, sentence, chunks, and document granularity, and it is adapted based on the task and the domain. In the current application the documents are really large, so it is necessary to split them. Different chunking strategies determine how a document is split to maintain meaningful context for each segment. The choice of strategy varies based on the application and the nature of the document collection; there is no universally optimal method. Given that effective chunking enhances retrieval accuracy, preserves context, and improves the efficiency of information processing, an important step of improving the RAG system involves selecting the appropriate chunking strategy and fine tuning the related hyper parameters, such as the dimension of the chunks and the chunks overlap for Recursive Character Splitting, and the threshold for semantic similarity in case of Semantic splitting, and also the number of chunks to extract in both cases, in order to give the right amount of context to the LLM, while maintaining efficiency of retrieval.

It is also interesting to analyze whether by providing only the necessary context, excluding irrelevant details, instead of providing larger context, the LLM's performance improves

or not. Additionally, it is crucial to take into account the fact that language models have a fixed content limit, a maximum number of tokens they can process, so for lengthy documents, as in this case, the splitting may become necessary also for technical reasons. While newer LLMs have increasingly larger token limits, reducing this constraint, it remains an important consideration based on the chosen models. By focusing on text splitting and tailoring the content to the specific task and to the LLM's capabilities, we ensure the input is both efficient and effective.

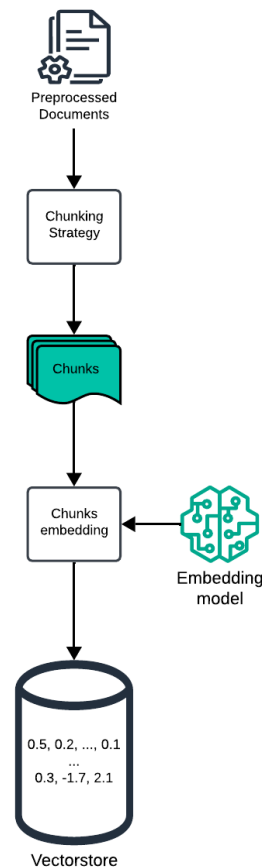


**Figure 4.7: Pipeline for vector database creation.** *The preprocessed documents are split into smaller chunks using a chunking strategy, and each chunk is then embedded resulting in vectors that capture the semantic meaning of the chunks, that are stored in the vectorstore, enabling efficient retrieval in the RAG pipeline.*

As shown in Figure 4.7, the preprocessed augmented documents are divided into manageable chunks, ensuring that the output of the chunking strategy is a collection of smaller text segments, ready for semantic embedding. In the subsequent step of the pipeline, the chunks are passed through the embedding model to generate the corresponding vector

representation. These vectors, that capture the semantic meaning of the text chunks in a high-dimensional space, are stored then in a vectorstore, an efficient vector database for fast retrieval, which serves then as the core retrieval mechanism for the RAG pipeline. Key decisions in this workflow include selecting the embedding model, determining the chunking strategy, and choosing the vectorstore. The choice of the embedding model and the vectorstore will be detailed in the following section, focusing now only on the chunking strategy. The chunking commandment is at the core of the methodology and the selection of the final chunking strategy.

---

*The Chunking Commandment: The goal is not to chunk for chunking sake, the goal is to get the data in a format where it can be retrieved for value later.[11]*

---

Ideally, the best chunking strategy is the one that keeps together the semantically related pieces of text, maintaining the right amount of context for an efficient information extraction. However, the definition of "semantically related" varies depending on the type of text and the application's purpose, adding complexity to the choice of the appropriate chunking strategy. Some of the main chunking strategies are detailed as follows.

The simplest chunking technique is the **Character Splitting** technique, which consists in simply dividing the text into N-character sized chunks regardless of their content or form[11]. This is a very simple and fast method, but doesn't take into account the structure of the text and can split apart not only phrases, but also words, cutting of the chunk whenever it reaches the number of N characters for each segment. In addition to adjusting the chunk size, it's important to consider the chunk overlap, which creates duplicate data across the chunks, preventing abrupt breaks in context, ensuring continuity from a chunk to another, assuming that the tail of chunk  $i$  will be the same as the head of the chunk  $i+1$ , preserving a smoother flow of information.

The Character Splitting method is a fixed-length chunking, dividing the text in uniform chunks, based on the predetermined number of N characters. As any technique, it has advantages and disadvantages. The main advantage is the fact that the implementation is fast and straightforward, and it maintains consistency across chunks, all of equal sizes, making the processing predictable and easier to fine tune. On the other hand, the disadvantages are mainly related to the fact that the method doesn't consider the natural language structure of the sentences and may split ideas and even words, leading to a potential loss of meaning, which diminishes the relevance of the retrieved information.

A better method for chunking documents is the **Recursive Character Text Splitting** technique, which solves the problem of not considering the structure of the document. The method uses a series of separators, employing a hierarchical approach, splitting the text using the multiple defined separators in a specific order, in order to find the most meaningful boundaries. The default separators, in order, are ["\n\n", "\n", " ", ""], separating first based on the paragraphs, then based on the new lines and the spaces, and only in the end by characters. The logic behind this method is that, once the paragraphs are split, the algorithm looks at the chunk size, and if it is too big, then it will split by the next separator and so on, until it can split the text according to the chunk size constraint. This technique has the main advantage of respecting the natural language structure in the text, maintaining the integrity of ideas among paragraphs and sentences as much as possible, and maintaining more coherent information, potentially reducing the need for a high N chunk size. This technique, unlike the previous method, is more complex and resource-intensive, especially for large texts.

A new experimental technique for approaching chunking, the **Semantic Chunking**, was first introduced by Greg Kamradt [11], given that the previous methods require a global chunking size, which may be too trivial for maintaining the meaning of segments within the document. This new chunking technique aims to extract semantic meaning from embeddings and then to assess the semantic relationship between these chunks iteratively, keeping together the texts that are semantically similar. The method is based on embeddings to compute the semantic similarity between parts of the text and in this approach, the main advantage is that it ensures the creation of coherent units of information, maintaining the semantic integrity of chunks. The disadvantage is that the method requires advanced techniques and it is resource-intensive in order to analyze the meaningful differences among sequent parts of text, especially in long documents; it also loses the overlaps among chunks, reducing the continuity.

The Semantic Splitting approach used in this thesis uses the following [11] "walk" method. The document is first divided into sentences. Then, starting with the first sentence, its embedding is compared to those of the subsequent sentences (second, third, etc.), looking for a break point. A break occurs when the embedding of the next sentence is too distant from the previous one, exceeding the predefined threshold for semantic continuity. Since analyzing single sentences may yield noisy results, the method considers groups of sentences instead. By evaluating the distances between groups and removing one sentence at a time during the walk, more reliable outcomes can be achieved. Thus, when the embedding distance of a sentence group exceeds the threshold, it signals the beginning of a new semantic section, separating the new group from the previous chunk.

The first experiments regard the Recursive Character Splitting Technique, using the simple preprocessing pipeline which doesn't remove the extra whitespace, given that it maintains the structure of the original text and as seen before, gives better overall results. For this kind of splitting, the chunk dimensions are fundamental, mainly the chunk size and the chunk overlap, which depend on the dataset and on the application.

| Chunk Size & Overlap   | ROUGE-1 | BERT F1 | SemScore | Observations                                                                                                                                                                                                                                                                                                                                 |
|------------------------|---------|---------|----------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Size 3000, Overlap 400 | 0.380   | 0.730   | 0.903    | Moderate scores, larger chunks retain context but may lose coherence, giving too much noise to the LLM and reducing the generation accuracy.                                                                                                                                                                                                 |
| Size 2000, Overlap 200 | 0.367   | 0.726   | 0.902    | Slightly lower scores; could be that in this case insufficient overlap may reduce continuity of context of the chunks passed to the LLM, not balancing correctly the chunk size and overlap.                                                                                                                                                 |
| Size 1500, Overlap 150 | 0.396   | 0.734   | 0.908    | Best performance; optimal balance between chunk size and overlap for this task, obtaining the best results over all the metrics.                                                                                                                                                                                                             |
| Size 1000, Overlap 100 | 0.385   | 0.728   | 0.904    | Decent performance, comparable to the results obtained with larger chunks, but having faster retrieval and more precise and concise context given to the LLM, using less input tokens and achieving same or even better results.                                                                                                             |
| Size 500, Overlap 50   | 0.362   | 0.725   | 0.906    | Trying to see how much the reduction of the chunks maintains the results comparable with the one obtained with larger context: considering the results over all metrics, the high fragmentation of information in this task leads to lower score, reducing the coherence and the usefulness of the context passed to the LLM for generation. |

Table 4.2: Hyper-parameter tuning of Chunk Size and Overlap for Recursive Chunking

The results of the hyper-parameter tuning experiments of chunk dimensions for recursive chunking reveal that the best results are achieved with a chunk size of 1500 and an overlap of 150, obtaining best performance across all metrics: a ROUGE-1 score of 0.396, BERT F1 of 0.734, and SemScore of 0.908. From the results, it can be supposed that this combination of dimensions is the one that creates an optimal balance between retaining sufficient context and avoiding excessive fragmentation and noise, making it the most suitable choice for this task. Larger chunk sizes, such as 3000, retain more context, but their extended length may reduce coherence and introduce noise, negatively affecting performance. On the other hand, smaller chunk sizes, such as 500, result in fragmented information that lowers coherence and diminishes the utility of the context provided to the language model.

After deciding the chunking dimensions best suited for Recursive Splitting other experiments aim to decide the best number of chunks to retrieve and pass to the LLM for the generation. The Table 4.3 gathers all the performed experiments in order to tune this hyper-parameter, not only for the recursive splitting, but also for the semantic one. The hyper-parameter tuning of the retrieval is performed using a simple dense retriever, available for vectorstores in the `langchain` library, which extracts the chunks that are semantically closer to the user query. The retriever converts the user query into a vector, which is compared against the vectors of the stored chunks using cosine similarity, obtaining a similarity score, which determines the most relevant chunks [21].

| Chunking Type | Dense k | ROUGE-1 | BERT F1 | SemScore |
|---------------|---------|---------|---------|----------|
| Recursive     | 2       | 0.377   | 0.725   | 0.905    |
| Recursive     | 4       | 0.396   | 0.734   | 0.908    |
| Recursive     | 6       | 0.382   | 0.732   | 0.905    |
| Recursive     | 8       | 0.392   | 0.734   | 0.907    |
| Recursive     | 10      | 0.397   | 0.733   | 0.907    |

| Chunking Type | Dense k | ROUGE-1 | BERT F1 | SemScore |
|---------------|---------|---------|---------|----------|
| Semantic      | 2       | 0.363   | 0.722   | 0.892    |
| Semantic      | 4       | 0.378   | 0.730   | 0.898    |
| Semantic      | 6       | 0.379   | 0.728   | 0.896    |
| Semantic      | 8       | 0.373   | 0.726   | 0.894    |
| Semantic      | 10      | 0.372   | 0.725   | 0.890    |

| Chunking Type | Dense k | ROUGE-1 | BERT F1 | SemScore |
|---------------|---------|---------|---------|----------|
| Recursive     | 4       | 0.396   | 0.734   | 0.908    |
| Semantic      | 4       | 0.378   | 0.730   | 0.898    |

**Table 4.3:** *Recursive and Semantic Character Text Splitting experiments and results with simple preprocessing, tuning dense k from 2 to 10. It compares the best results of the two chunking techniques in the last table, confirming  $k = 4$  as most effective value for retrieval.*

From these results, several conclusions can be derived. Analyzing the results of Recursive Character Text Splitting, the derived conjecture is that smaller chunks may provide the most relevant context, but often lack sufficient surrounding information, which increases the likelihood of hallucinations and repetitive answers, therefore limiting the answer quality. On the other hand, larger chunks can slightly reduce focus and performance if the context is not well-balanced, or they may achieve similar results to more balanced configurations, thanks to the LLM’s ability to filter out noise. Since the results obtained with a higher number of extracted chunks are comparable to those achieved with  $k = 4$ , the additional context does not yield significant improvements and can even reduce efficiency by passing more noise and less relevant information. Thus, to balance context, response quality, and accuracy while minimizing noise, the configuration with  $k = 4$  is identified as the most effective for recursive character text splitting with a chunk size of 1500 and an overlap of 150, which were the configurations that gave the best results in the previous analyses and hyper-parameter tuning of chunking dimensions.

Analyzing, as well, the Semantic Chunking strategy, it can be observed a general performance reduction compared to the results of the recursive chunking. This outcome may have two main reasons:

1. **Variability in chunk sizes:** given that semantic chunking adapts the chunks based on the meaning, the procedure leads to chunks of very different sizes, which makes it harder to define the best number of chunks to retrieve, the hyper-parameter  $k$ , as the system may extract content with significantly varying lengths. If smaller chunks are retrieved, the model may need more of them to form a complete understanding, while larger chunks might convey more information but result in redundant or irrelevant retrievals. Trying values of  $k$  from 2 to 10, the only thing that can be noticed with confidence is that using  $k = 2$  gives the worst performances, risking to provide not enough context to the LLM. For the other experiments, there aren’t significant differences among the obtained metrics, noticing only a small improvement for  $k = 4$ , which is considered the best overall result;
2. **Bias intrinsic to the creation of the evaluation set:** the expectation was that semantic chunking would outperform recursive chunking because it groups contextually similar content into cohesive chunks, despite the challenges of tuning associated hyper-parameters (such as  $k$  and the similarity threshold for embeddings). However, the results did not meet this expectation and a plausible explanation for this discrepancy is the presence of bias in the evaluation process, which appears to favor the recursive chunking approach due to the way the evaluation set was created. The evaluation set, made of question-answer pairs, was synthetically generated using

an LLM prompted to create pairs based on some randomly adjacent chunks produced by the recursive chunking technique, so this process potentially disadvantages the semantic chunking strategy, which generates the answer based on more varying chunks, less aligned with the structure of the evaluation set. Details on the testing framework and the methodology for creating the evaluation set will be further explained in [Chapter 5](#). However, it is important to acknowledge this potential bias when interpreting the results, which seem to favor recursive chunking, possibly due to closer alignment between the extracted chunks and those used to generate the evaluation set, produced with the same technique but different hyper-parameters.

To address the possible bias inherent to the evaluation set, a new mixed evaluation set was created to compare the two best chunking strategies, with the chosen configuration of hyper-parameters. The mixed evaluation set is also composed of around 100 question-answer pairs, in order to maintain comparability with previous results. The important aspect of the new mixed evaluation set is that it is evenly divided: half of the pairs are derived from chunks generated using recursive splitting, while the other half is obtained using chunks created through semantic splitting. Thus, creating the evaluation set balancing both techniques should reduce the potential bias, obtaining more reliable results when comparing the different chunking techniques. In the following table, [Table 4.4](#), the best configurations of the two previously top-performing chunking techniques are evaluated on the mixed evaluation set.

| Chunking Type | Dense k | ROUGE-1 | BERT F1 | SemScore |
|---------------|---------|---------|---------|----------|
| Recursive     | 4       | 0.395   | 0.732   | 0.912    |
| Semantic      | 4       | 0.405   | 0.734   | 0.906    |

[Table 4.4](#): *Comparison of Best Recursive and Semantic Chunking strategies on a mixed evaluation set, to reduce bias and obtain more reliable and robust results*

Given that the semantic chunking achieves a slightly higher ROUGE-1 score, the chunks extracted with this technique seem to steer the LLM generation in a way that the generated content has a more accurate word overlap with the desired answers. This indicates that the generated answers better match the specific wording of the target answers when using semantic splitting. On the other hand, while the BERT-F1 scores are almost identical, the semantic similarity score computed with SemScore gives better results for the recursive chunking, indicating that this approach better preserves semantic coherence and relevance in the generated answers. This suggests that recursive splitting slightly facilitates deeper alignment with the underlying meaning of the target responses, even if it has lower lexical overlap.

Semantic chunking methods, especially those based on clustering or break-point detection, have significantly higher computational overhead compared to simpler approaches like recursive or fixed-size chunking. While semantic chunking provides some benefits, its performance improvements are inconsistent across tasks like evidence retrieval and answer generation, and often do not justify the additional costs [22]. In this thesis, the semantic chunking explained before is categorized as one based on break-point detection, and while it provides some benefits, having a slightly improved lexical alignment given the ROUGE-1 score, this improvement is too small and it doesn't seem to outweigh the practical drawbacks of higher latency and resource usage.

In the context of a chatbot assistant for customer support in an ERP system, aspects like accuracy, speed, and efficiency are crucial, so in choosing the best chunking strategy, one cannot focus only on the quality of the generation results, but it is necessary to also consider the system's overall performance under real-world constraints. Thus, when considering the application context and the computational efficiency, together with the results across the considered metrics, the recursive chunking is the preferred choice for the chatbot, given that it achieves comparable results to the ones of the semantic chunking in most metrics and it even surpasses it in preserving the semantic coherence, all while having much lower computational cost and ensuring faster response times, making it more practical for real-time interactions and more suitable for the task.

The impact of the chunking strategy is often intertwined with other critical components, such as the quality of the embeddings. The next section will address the choice of embedding models, which significantly influence the retrieval quality and the similarity between chunks and user queries. Additionally, the vector-store used to store the chunk embeddings will be discussed, given that it has a crucial impact on the efficiency, the speed, and the scalability of the RAG system. Together, all these elements, the embedding model, the vector-store, and the chunking strategy, directly affect the retrieval and the generation performance of the system, and their intertwined connection makes it really hard to optimize these components to achieve the best implementation for the application and the dataset at hand. It is also important to remember that having a ground-truth dataset with question-answer pairs would provide much more accurate evaluations than the synthetic evaluation set the system is currently tested on. A future crucial development consists of curating a reliable evaluation set over time, which would represent a more robust benchmark for the evaluation of the retrieval and the generation steps of the RAG pipeline, in order to get insights and further improve the system results as needed.

#### 4.2.5. Embedding model and Vector-store creation

After chunking the data, as shown in Figure 4.7, it is necessary to convert the chunks of text into vector representations using an embedding model, which converts the text into a vector representation that captures the semantic meaning of the chunk. The chunks embeddings can be then stored in a vectorstore, allowing for fast and accurate retrieval of the most relevant chunks based on the semantic similarity to the user's query, which will be also transformed using the same embedding model used for the chunks, in order to have a meaningful comparison [19].

In choosing the embedding model, different aspects were taken into consideration in order to get optimal performance for the specific use case, considering the best embedding aligning with the data at hand. The factors that need to be taken into consideration when choosing the embedding model are: the model's maximum token limit to avoid truncating inputs and losing important information; the model size, to strike a balance between performance and efficiency; and the characteristics of the model itself, as its performance may vary depending on the training data and the languages it has been exposed to during training [19]. Many off-the-shelf embedding models were experimented for the task in order to find the best one. The biggest struggle was finding a model that could handle multilingual data effectively, particularly the Italian language, which is the only language used in this chatbot. In fact, despite the popularity and the massive use of embeddings, the existing models often lack versatility, especially the open source ones. Most are mainly tailored for English, offering fewer optimal choices for other languages, and many can only handle short inputs, which limits their utility in processing longer text chunks [20].

The embedding models were selected from the famous benchmark called MTEB (Massive Text Embedding Benchmark), verifying each time compatibility with the Italian language and a sufficiently large token input limit. For the task at hand, the best results were obtained with the BAAI/bge-m3, the M3-Embedding model, a multi-linguality, multi-functionality and multi-granularity model. It supports semantic retrieval for over 100 languages, including Italian, for which it demonstrates strong quality and performance; and it is capable of processing text of different granularities, from short sentences to long documents up to 8192 input tokens [20].

After applying the embedding model, the chunk embeddings are saved in a vector-store, in this case in the Facebook AI Similarity Search (FAISS) vector database. Vector databases play a crucial role in the retrieval part of the RAG by efficiently storing and retrieving the vector embeddings, and they are specialized in retrieving vectors based on similarity,

computing it quickly and accurately. The vector representations of the chunks are stored together with the chunk's text into the vector database, which includes a vector index that uses different algorithms to organize and map the vector embeddings in a way that optimizes search efficiency. At query time, a user's request is embedded into a vector, and the database leverages the vector index to find the most similar chunk vectors, returning the corresponding original text chunks [23].

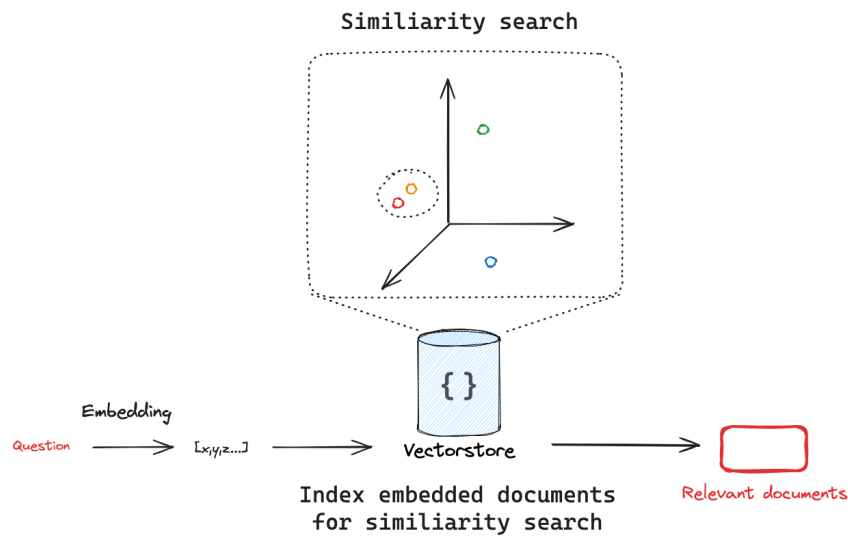


Figure 4.8: **Vectorstore and similarity search [25]:** search over unstructured data to retrieve relevant chunks for user query based on semantic similarity.

The vector store for this application is implemented using the FAISS library in `langchain`, a library for efficient similarity search and clustering of dense vectors, allowing a fast and accurate retrieval of relevant chunks based on the user question. `Langchain`'s standardized interface for various vector store integrations also allows easy switching and experimentation with different databases [24]. The FAISS vector store was chosen for its ability to handle large-scale databases, crucial for the RAG system's extensive knowledge base; and also for its indexing algorithms that ensure fast and customizable retrieval, easily integrated into the RAG pipeline. Given its satisfactory performance, other vector databases were not explored, but they can be easily swapped if needed. Another important aspect is that FAISS can be combined with traditional retrieval methods, such as keyword-based retrieval, to create hybrid systems that mix semantic similarity with exact matches, enhancing retrieval precision. This hybrid approach will be integrated into the Naive RAG system, introducing a more advanced retrieval technique that, as explained in the following section, provides slightly better overall performance on this specific task.

### 4.3. Naive RAG implementation with improvements

This chapter presents a Naive RAG pipeline, without any of the previous optimizations, and compares this baseline with two improved implementations. The first enhanced version integrates all the best techniques and configurations identified so far, while the second further refines the process, especially the retrieval step of the process. Specifically, different Information Retrieval (IR) techniques are explored in order to choose the best one for the task. Another aspect that will be explained in this chapter regards the optimized model prompt, already integrated in the results presented so far in this thesis, and improved, following an iterative prompt engineering procedure. The improved implementations will be evaluated against the baseline using multiple performance metrics, the same used so far, the ROUGE-1, BERT-F1, and SemScore, to report an end to end evaluation of the pipeline and display the impact of these techniques on the final results.

#### 4.3.1. Baseline Naive RAG implementation

The Naive RAG provides a simple, unoptimized version of the pipeline, quickly implemented, without extensive experimentation, in order to obtain a simple model to use as baseline for further experiments, analyzing iterative improvements and the impact of the chosen techniques. This basic Naive RAG implementation has the following features:

1. **No augmentation and preprocessing:** being a simple baseline it uses the raw text content extracted from the PDFs, without any data augmentation and data preprocessing, leaving redundancies, inconsistencies, and unwanted patterns in the data, having much more noise than in the following improved implementations;
2. **Chunking with default parameters, without hyper-parameter tuning:** the raw text is split using recursive chunking with chunk size of 3000 tokens and chunk overlap of 500 tokens, chosen arbitrary, prioritizing large context size with the idea of maintaining as much as possible relevant information;
3. **Basic retrieval mechanism:** the retrieval step relies on dense retrieval, embedding the user query with the same embedding model used for the chunks embeddings and extracting the most similar chunks based on the cosine similarity score; in this case, the number of retrieved chunks was not fine tuned and it was simply set to  $k = 6$ ;
4. **Simple prompt structure, without any prompt engineering:** the LLM is prompted with a simple but clear input, that combines the retrieved chunks with the user query and basic task instructions; while functional, this prompt lacks any optimization, leading to potential ambiguities and noise in the responses.

This baseline has some factors in common with the following improved versions, in particular the embedding model and the vector store, using the BAAI/bge-m3 embedding model and the FAISS vector database, explained in the previous chapter.

The baseline pipeline's workflow, in Figure 4.9, represents the main components of a RAG pipeline, emphasizing the two main data flows. The first flow creates the knowledge base: the documents are split into manageable chunks, embedded and then stored in the chosen vectorstore. The second flow begins with the user query, which is embedded and compared to the chunks in the vectorstore using the chosen retrieval technique, extracting the most relevant chunks, which are included in the prompt for the LLM's final answer generation. These two processes are present in all RAG pipelines, with each configuration and technique influencing the final results.

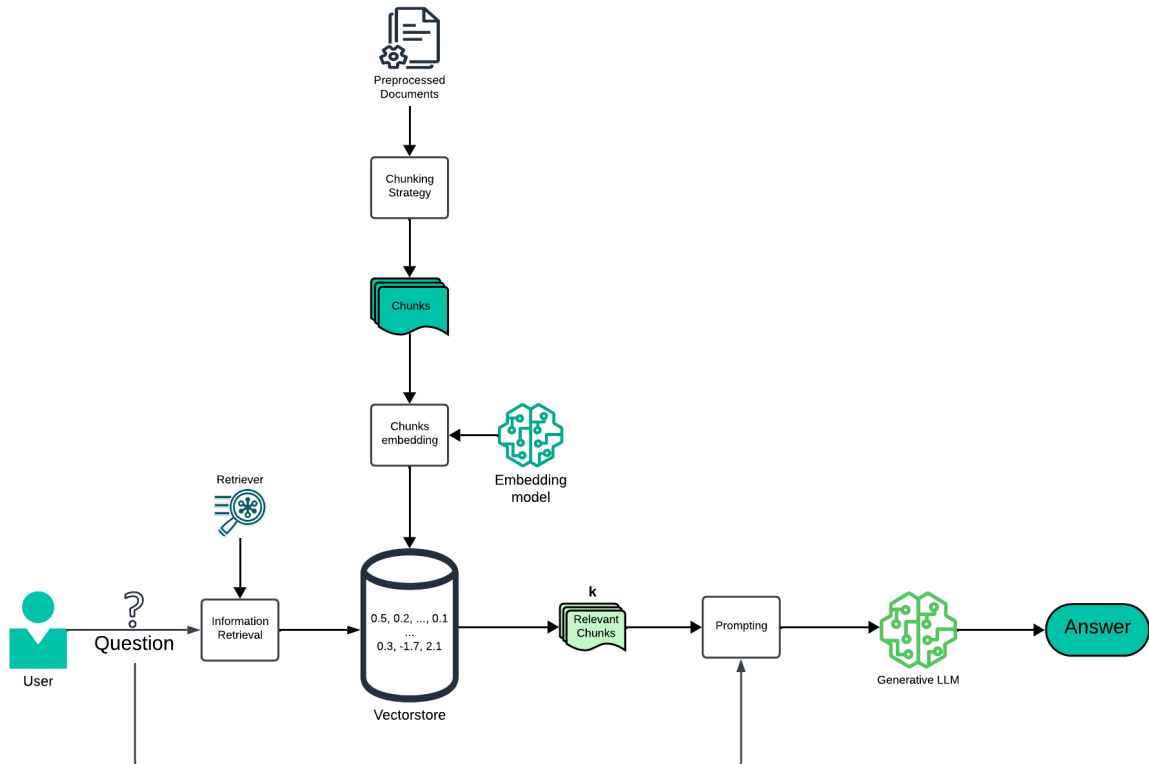


Figure 4.9: **Basic RAG pipeline:** *given the user question, the chosen information retrieval technique is used to extract from the vectorstore the most relevant chunks, then combined with the initial query and instructions to guide the LLM's response.*

### 4.3.2. Improved Naive RAG implementation

The improved Naive RAG pipeline integrates all the techniques discussed in the previous chapter, addressing the limitations of the baseline implementation, enhancing mainly the

indexing, but also the retrieval and the generation part of the pipeline using simple and quickly implemented techniques. This implementation is at the basis of the final results presented in the previous chapter, thus, it integrates all the elements discussed before:

1. **Data augmentation and preprocessing:** the raw textual content is augmented with image descriptions using Gemini and then preprocessed using the simple data cleaning pipeline without white-spaces normalization, in order to maintain the structured of the original documents;
2. **Optimized recursive chunking:** the recursive chunking, preferred with respect to the semantic one given the previous results and computational trade-off, was optimized through hyper-parameter tuning; the optimal configuration is achieved with chunk size 1500 and chunk overlap 150, as extensively detailed in the previous sections;
3. **Optimized retrieval:** the improved pipeline maintains the embedding model and vector store, explained so far, and uses as information retrieval a dense retrieval technique, but introduces the fine tuning of the retrieval parameter  $k$ , selecting as optimal number of chunks to extract  $k = 4$ , ensuring sufficient context without overloading the language model with unnecessary details;
4. **Simple prompt engineering:** the prompt structure was refined to include more explicit task instructions and solve some problems identified in the prompt development process, enhancing it to improve the clarity and the relevance of the responses.

The improvements introduced in this implementation are summarized in Table 4.5, which compares the baseline and improved version of the Naive RAG pipeline. The results clearly demonstrate that the improved Naive RAG pipeline outperforms the baseline across all metrics. The enhancements in data augmentation, chunking, retrieval, and prompt engineering, collectively contributed to significant improvements in ROUGE-1, BERT F1, and SemScore metrics.

| Implementation              | ROUGE-1 | BERT F1 | SemScore |
|-----------------------------|---------|---------|----------|
| Naive RAG, simple baseline  | 0.307   | 0.699   | 0.871    |
| Naive RAG with improvements | 0.396   | 0.734   | 0.908    |

**Table 4.5:** Comparison between simple Naive RAG and Naive RAG with the previously explained improvements: advanced indexing using data augmentation and simple preprocessing, selecting the chunking strategy with associated hyper-parameter tuning; and improved retrieval by fine tuning the number of extracted chunks to pass to the LLM.

Additional enhancements to the Naive RAG pipeline focus on further improving retrieval, selecting the most effective IR technique. This chapter will provide an overview of some of the experiments conducted in order to identify the optimal retrieval technique; and will also delve deeper into the process behind the development of the optimized prompt, already integrated into the previous results. The following sections will elaborate on these techniques and report the final results for the second version of the improved Naive RAG.

## Information Retrieval Technique

The implementations considered so far rely on dense retrieval, using semantic similarity in order to extract the most relevant chunks for the user query. However, some other retrieval techniques can be explored to improve the retrieval accuracy, passing more relevant content to the LLM for the generation. Three main retrieval paradigms are considered here:

1. **Sparse retrieval:** this traditional search method relies on exact or partial string matching for retrieval, a keyword-based search, whose main limitation is that it completely lacks semantic understanding; commonly used techniques are TF-IDF and BM25, both of which are algorithms that perform lexical search based on word frequency; if a word appears frequently both in the user query and in a chunk, then the chunk might be considered relevant to answer the query [26];
2. **Dense retrieval:** is based on semantic similarity, computing, for example, the cosine similarity among dense vector representations generated from embeddings, which capture the meaning and context of words and phrases; this enables to find related content, even when different wording is used for the same concept, surpassing the limitation of sparse retrieval;
3. **Hybrid retrieval:** combines sparse and dense retrieval, balancing the impact of the two strategies; an ensemble retriever can be used to combine the two techniques with a linear weighted score, that determines the final retrieval results and the method's impact on these results.

When ensembling multiple retrievers, it is necessary to decide how to combine the search results. This motivates the use of filters and reranking mechanisms, which take the output of multiple retrievers and combine only the best of them, weighting the content differently, using more sophisticated algorithms, such as Reciprocal Rank Fusion (RRF) [26]. This advanced technique will be further explained in the following chapter of the Advanced RAG, comparing the results obtained here with the ones achieved using RRF, additional filtering, and other sophisticated approaches.

After many experiments and hybrid retrieval fine-tuning, considering both retrieval and end to end quality assessment, the best obtained configurations are the ones summarized in the following table:

| Retriever Strategy      | Sparse | Dense  | ROUGE-1 | BERT F1 | SemScore |
|-------------------------|--------|--------|---------|---------|----------|
| Sparse Retriever BM25   | 4      | 0      | 0.351   | 0.721   | 0.889    |
| Sparse Retriever TF-IDF | 4      | 0      | 0.379   | 0.727   | 0.901    |
| Dense Retriever         | 0      | 4      | 0.396   | 0.734   | 0.908    |
| Hybrid Retriever BM25   | 4(0.4) | 4(0.6) | 0.396   | 0.734   | 0.907    |
| Hybrid Retriever TF-IDF | 4(0.2) | 2(0.8) | 0.397   | 0.737   | 0.907    |

*Table 4.6: Performance results of different retrievers, evaluated using the downstream metrics that best align with human judgment in the specific scenario.*

It can be observed that the sparse retrievers, both BM25 and TF-IDF, while competitive on some metrics, generally perform worse than the dense retrieval; and it seems that, in this scenario, the BM25 struggles more when extracting relevant chunks. In general, BM25 is considered an improvement over TF-IDF, as it normalizes by document length and captures term-frequency saturation. For this specific dataset and task, though, a more direct and linear weighting of term frequencies and inverse document frequencies, as in TF-IDF, results in better performance.

From the results, also, emerges that the dense retrieval strategy on its own, gives very strong performances, comparable to the ones of the hybrid retriever configurations. In principle, hybrid retrieval is more robust to variations and compensates the weaknesses of each approach, merging both semantic relevance and exact keyword extraction. This advanced retrieval technique gives almost the same results as the dense retriever, surpassing it slightly only in the hybrid configuration that incorporates TF-IDF. The hybrid method leverages the efficiency of sparse retrieval, while preserving semantic meaning, allowing it to scale better to larger datasets, reduce computational cost, and achieve marginally better results.

Since, in this scenario, the data and the user queries may change with new software versions and updates, a more robust retrieval strategy is preferable. Given that hybrid retrieval has a more stable behavior in case of changes, since it doesn't depend only on one representation method (lexical or vector-based), it will be the one adopted for the next stages of pipeline improvement. Therefore, hybrid retrieval is a better choice than dense retrieval, and in the following experiments, building on these findings, other techniques will be explored to further enhance the quality of the hybrid retrieval. In the following chapter, the RRF algorithm will be applied on top of the RAG-pipeline with hybrid retrieval, aiming to refine the final rankings, to improve the system performance.

## Prompt engineering

The main idea when prompting a LLM is to give clear and specific instructions. In this case, the prompt will contain the content of the chunks extracted during retrieval, the original question provided by the user, and detailed instructions to clarify the context and define the task. Following this idea of clarity and specificity, the initial prompt explicitly details the task and the role of the system, which has to act as an AI assistant that effectively addresses user queries. It is also specified the required response style, the level of detail, and the main focus of the assistant. The input prompt outlines every step necessary to complete the task: consider the user query, evaluate the retrieved content, and generate a response based only on the extracted information.

---

*itaPrompt = "Comportati come un assistente che risponde alle domande del cliente. Rispondi alla domanda basandoti solo sui seguenti documenti: {context}. Rispondi in modo conciso e chiaro, spiegando passo passo al cliente le azioni necessarie da effettuare. Se possibile, dai indicazioni dettagliate al cliente, su come risolvere il problema o effettuare l'azione desiderata. Domanda relativa al software Panthera: {question}."*

*engPrompt = "Act as an assistant responding to customer questions. Answer the question based only on the following documents: {context}. Respond concisely and clearly, explaining step by step the actions the customer needs to take. If possible, provide detailed instructions to the customer on how to solve the issue or perform the desired action. Question related to the Panthera software: {question}."*

---

Despite both Italian and English prompts being presented in this chapter, after analysis, it was decided that using Italian prompts is preferable, as they yield better results. This is likely because, using the Italian prompt ensures more consistency in the input by having a single language, given that both the retrieved content and the user query are in Italian.

To improve the prompt, an iterative procedure was followed, consisting of three main steps: prompt creation, experimental evaluation, and error analysis. Based on human evaluation and analysis of the model's responses, the prompt is adjusted, tested, and analyzed again, proceeding with this cycle until satisfactory results are obtained. Given that it doesn't exist a perfect prompt or a bunch of perfect prompts in absolute, the input must be tailored to the specific task and domain through extensive evaluation. Following this strategy for prompt development, first, a batch of test questions was selected; after analyzing the results of the previous simple prompt on this batch of questions, some issues were identified, particularly the fact that sometimes the model answers to out-of-scope questions, which are unrelated to the documentation and the software domain.

The prompt was adjusted to ensure the model avoids answering to irrelevant questions.

---

*itaPrompt = "Comportati come un assistente che risponde alle domande del cliente. Rispondi alla domanda basandoti solo sui seguenti documenti: {context}. Rispondi in modo conciso e chiaro, spiegando passo passo al cliente le azioni necessarie da effettuare. Se possibile, dai indicazioni dettagliate al cliente, su come risolvere il problema o effettuare l'azione desiderata. **Se non sai rispondere alle domande relative alla documentazione, rispondi soltanto che non detieni queste informazioni e che è necessario contattare l'assistenza del helpdesk. In caso di più domande rispondi solo a quelle inerenti alla documentazione e rimani a disposizione per altre domande sull'argomento, specificando, invece, che le altre domande non sono state trovate pertinenti in questo contesto.** Domanda relativa al software Panthera: {question}."*

*engPrompt = "Act as an assistant responding to customer questions. Answer the question based only on the following documents: {context}. Respond concisely and clearly, explaining step by step the actions the customer needs to take. If possible, provide detailed instructions to the customer on how to solve the issue or perform the desired action. **If you do not know how to answer questions related to the documentation, simply state that you do not have this information and that it is necessary to contact the help desk assistance. If there are multiple questions, answer only those related to the documentation and remain available for other questions on the topic, specifying that other questions are not relevant in this context.** Question related to the Panthera software: {question}."*

---

With this more complex prompt the previous problem is solved, but another one is created, the fact that, now, the model doesn't answer to some questions it should be answering to. To mitigate this problem, the prompt was revised making it more clear and concise, in order to not confuse the model, and filter out only irrelevant questions. Additionally, it was also observed that even if the user query is not about troubleshooting errors, but about understanding some concepts, the model treats everything as a problem, offering step by step solutions, instead of a clear explanation, and this behavior is not desirable.

---

*itaPrompt = "Comportati come un assistente che risponde alle domande del cliente. Rispondi alla domanda basandoti solo sui seguenti documenti: {context}. Rispondi in modo conciso e chiaro, spiegando passo passo al cliente le azioni necessarie da effettuare. Se possibile, dai indicazioni dettagliate al cliente, su come risolvere il problema o effettuare l'azione desiderata. **Quando spieghi che cosa è o cosa significa un certo elemento richiesto, non parlarne come se fosse un problema. In caso di più domande rispondi solo a quelle inerenti alla documentazione e rimani a disposizione per altre domande sull'argomento, specificando, invece, che le altre domande non sono state trovate pertinenti in questo contesto.** Domanda relativa al software Panthera: {question}."*

*engPrompt = "Act as an assistant responding to customer questions. Answer the question based only on the following documents: {context}. Respond concisely and clearly, explaining step by step the actions the customer needs to take. If possible, provide detailed instructions to the customer on how to solve the issue or perform the desired action. **When explaining what a certain requested element is or means, do not refer to it as if it were a problem. If there are multiple questions, answer only those related to the documentation and remain available for other questions on the topic, specifying that other questions are not relevant in this context.** Question related to the Panthera software: {question}."*

---

Following these refinements, the prompt effectively resolved all identified issues. After a further analysis of the generated answers, they were deemed of good quality, but suffering from redundancy, having too many variations of the same concept. This step of the prompt development approach, thus, aims to minimize unnecessary repetitions.

---

*itaPrompt = "Comportati come un assistente che risponde alle domande del cliente. Rispondi alla domanda basandoti solo sui seguenti documenti: {context}. Rispondi in modo conciso e chiaro, spiegando passo passo al cliente le azioni necessarie da effettuare. Se possibile, dai indicazioni dettagliate al cliente, su come risolvere il problema o effettuare l'azione desiderata. **Evita troppe ripetizioni nella risposta fornita.** Quando spieghi che cosa è o cosa significa un certo elemento richiesto, non parlarne come se fosse un problema. In caso di più domande rispondi solo a quelle inerenti alla documentazione e rimani a disposizione per altre domande sull'argomento, specificando, invece, che le altre domande non sono state trovate pertinenti in questo contesto. Domanda relativa al software Panthera: {question}."*

*engPrompt = "Act as an assistant responding to customer questions. Answer the question based only on the following documents: {context}. Respond concisely and clearly, explaining step by step the actions the customer needs to take. If possible, provide detailed instructions to the customer on how to solve the issue or perform the desired action. **Avoid too many repetitions in the provided answer.** When explaining what a certain requested element is or means, do not refer to it as if it were a problem. If there are multiple questions, answer only those related to the documentation and remain available for other questions on the topic, specifying that other questions are not relevant in this context. Question related to the Panthera software: {question}."*

---

This final version of the prompt, obtained after extensive testing, balances relevance, clarity, and conciseness, while solving all the previous issues, obtaining satisfactory results. Despite these improvements, one limitation remains: to answer some questions, the model sometimes specifies low-level configuration aspects that are unavailable to end users, as such modifications can be performed only by the company's help desk. It was not possible to solve this problem, given that in the documentation, it is not specified

what actions the user has at his disposal and what configurations need more permissions to be modified. This issue cannot be solved using only prompt engineering; it requires a documentation enhancement to delineate user capabilities and company-controlled low-level configurations. This will be an important aspect to take into consideration for future developments and system improvements.

Nonetheless, for now, the obtained prompt represents the best possible outcome, but further improvements can be explored in the future, implementing one-shot and few-shot prompting, selecting the most effective examples with the guidance of human experts.

#### 4.4. Advanced RAG implementation

The improved Naive RAG pipeline focused mainly on improving the indexing and the data ingestion, while also refining the retrieval and generation processes. The Advanced RAG pipeline builds upon the best results obtained so far, aiming to address some critical challenges identified in the analysis. This implementation prioritizes the improvement of both retrieval and final generation, introducing cost-efficient advanced techniques.

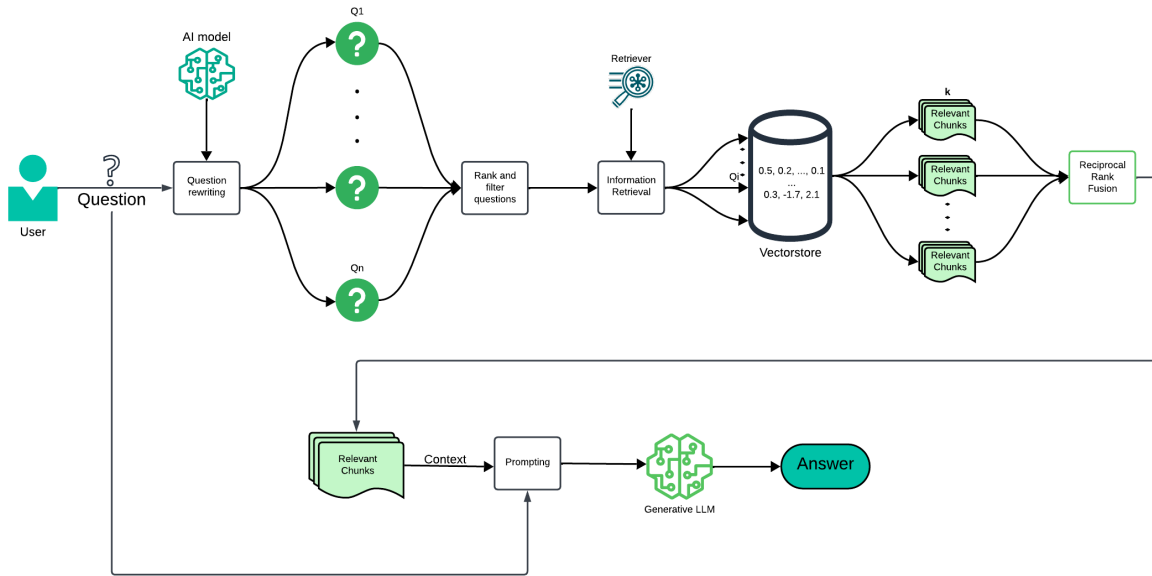


**Figure 4.10: Advanced RAG:** building upon the previous pipeline, while maintaining the same flow of information and crucial steps, this new RAG implementation incorporates other advanced techniques, specifically, the query rewriting step and the RRF algorithm which improves the relevance of the extracted information. The advanced RAG aims to an optimized generation based on more effective context, an improved prompt, and the selected LLM, in order to have a high-quality final answer while balancing cost and performance.

Techniques such as query rewriting and Reciprocal Rank Fusion (RRF) are employed to improve the quality and the ranking of the extracted chunks, while for the improvement of the generative part different models will be evaluated together with some optimized prompts, to achieve the most high-quality responses.

#### 4.4.1. Query rewriting and Reciprocal Rank Fusion

One of the main challenges in Naive RAG is that it relies directly on the user’s original query for the retrieval, which is often noisy and imprecise. An important step in improving the pipeline is to focus on the pre-retrieval process by preprocessing the query submitted by the user in order to enhance the formulation, obtaining a clearer and more precise question to use for retrieval [33]. Going beyond basic query preprocessing, the advanced technique considered here consists in query expansion, expanding a single query into multiples queries, employing prompt engineering to transform the query using LLMs, a mechanism designed in order to obtain variations of the original question that can explore different aspects of the query, and retrieve a broader variety of relevant chunks, while also clarifying the user intent when necessary. Given this step, to remove the noise of the original question, only the rewritten questions will be then used in parallel for the chunks retrieval, having a Rewrite-retrieve-read mechanism, aiming to enhance the retrieval effectiveness by optimizing the likelihood of getting the desired results [15].

This approach focuses on the adaptation of the search query, addressing not only the noise present in the original question, but also the gap between the input text and the knowledge to retrieve. The used approach is simple, yet effective, asking an LLM to rewrite the queries in order to have an improved search of the needed information. The prompt design is not limited to clarifying the context, defining the task, and instructing the model, but it also leverages advanced techniques, like one-shot and few-shot, to encourage the LLM to think and give as output multiple refined queries, used for the search [34]. The following sections detail some of the performed experiments, focusing on choosing the best prompt formulation, as research has shown that the phrasing of the prompt lines and the selection of demonstrative examples have an impact on query rewriting performance, but there is still going to be a limit to this technique’s effectiveness. Additionally, different models are evaluated, in order to choose the one which yields the best results and gives high-quality rewritten queries to improve the retrieval, given that this first step of the pipeline is critical and influences all the subsequent processes of the RAG.

The first set of experiments explores zero-shot prompting, comparing the effectiveness of Italian and English prompts across different models, to identify the best combination for

the task. Once the optimal model is selected, further prompt engineering is performed, adding positive and negative examples to encourage deeper reasoning in the LLM, and the results are analyzed balancing token usage and performance improvement.

#### Zero-shot prompting, comparing the effectiveness of Italian and English prompts across different models for question rewriting.

The model is instructed to assist the users by rewriting only the user’s questions related to the software, filtering out questions that are outside the domain. In this phase some experiments were conducted to set the number of rewritten questions to generate, deciding upon the generation of 5 rewritten questions, that maintain the original intent while exploring different formulations of the concept to enhance retrieval. In **Appendix B** more details can be found on the actual prompting process and the best obtained formulations, to ensure clear query rewriting while disregarding irrelevant or ambiguous requests.

The following table shows the results of a comparative analysis of multiple LLMs using the optimized zero-shot prompting in both Italian and English, evaluating the results based on the three metrics used for this entire thesis, ROUGE-1, BERT-F1 and SemScore:

| Model             | Language   | BERT F1       | ROUGE-1       | SemScore      |
|-------------------|------------|---------------|---------------|---------------|
| llama3.2          | Ita        | 0.7370        | 0.3348        | 0.8696        |
| llama3.2-instruct | Ita        | 0.7242        | 0.3041        | 0.8591        |
| llama3.2          | Eng        | 0.7416        | 0.3632        | 0.8743        |
| llama3.2-instruct | Eng        | 0.7360        | 0.3534        | 0.8614        |
| gpt-3.5-turbo     | Ita        | 0.7571        | 0.4098        | 0.9160        |
| gpt-3.5-turbo     | Eng        | 0.7480        | 0.3860        | 0.8818        |
| <b>gpt-4o</b>     | <b>Ita</b> | <b>0.7598</b> | <b>0.4277</b> | <b>0.9207</b> |
| gpt-4o            | Eng        | 0.7606        | 0.4281        | 0.9171        |
| gemini-1.5-pro    | Ita        | 0.7314        | 0.3635        | 0.8187        |
| gemini-1.5-pro    | Eng        | 0.7551        | 0.4053        | 0.8412        |

*Table 4.7: Performance comparison of different LLMs on Zero-shot prompting on small test set: GPT-4o model with Italian prompt is the combination that yields the best results.*

From the obtained scores, it can be observed that **GPT-4o** is the model that achieves the highest score across all metrics, achieving strong results for both Italian and English prompts. However, the Italian prompts are preferred, as they perform well and maintain consistency with the language used in the application. Since question rewriting is a critical step in the pipeline, where a robust behavior is necessary in order to avoid introducing noise in the early stages of the process, selecting the most accurate and semantically reliable model is crucial, thus, **GPT-4o** is the model of choice for this implementation.

To further explore the potential improvement of the generation, a comparative analysis of different prompting techniques was performed, evaluating not only zero-shot, but also one-shot and few-shot prompting on the best model found at the previous step: GPT-4o with Italian prompt.

**Comparative analysis of advanced prompting with GPT-4o and Italian prompt.**

| Prompting  | BERT F1 | ROUGE-1 | SemScore |
|------------|---------|---------|----------|
| Zero-shot  | 0.7598  | 0.4277  | 0.9207   |
| One-shot   | 0.7583  | 0.4278  | 0.9112   |
| Few-shot-2 | 0.7669  | 0.4368  | 0.9174   |
| Few-shot-3 | 0.7626  | 0.4410  | 0.9120   |
| Few-shot-4 | 0.7656  | 0.4428  | 0.9157   |
| Few-shot-5 | 0.7658  | 0.4468  | 0.9154   |
| Few-shot-6 | 0.7665  | 0.4437  | 0.9156   |
| Few-shot-7 | 0.7620  | 0.4406  | 0.9206   |
| Few-shot-8 | 0.7624  | 0.4370  | 0.9212   |
| Few-shot-9 | 0.7622  | 0.4503  | 0.9203   |

*Table 4.8: Comparison of Zero-shot, One-shot and Few-shot prompting using GPT-4o with Italian prompt: introducing few-shot prompting leads to an upgrade, but adding too many few-shot examples does not result in consistent improvements across the evaluated metrics, having excessive input tokens without significant performance enhancements.*

Transitioning from zero-shot to one-shot prompting does not appear to improve the results, while using a few shot with two examples, a positive and a negative one, there are some improvements in both BERT-F1 and ROUGE-1 score, also maintaining a high Sem-Score, which suggests that adding a few well-chosen examples helps the model generalize better and improves its performance on the task. However, trying to explore how many examples can benefit to the model generalization, it can be observed that in this specific case, increasing the examples does not yield consistent improvement across the evaluated metrics, so adding more examples does not continuously improve the performance, reaching a limit. Thus, the obtained results support the hypothesis that after N examples, additional examples do not provide substantial new information [35].

For this application, the findings suggest that few-shot with two examples provides the best cost-benefit trade-off, obtaining some improvements with respect to the zero-shot and one-shot prompting with minimal additional cost. Adding more examples in the few-shot prompting can lead to some improvements, but not consistent over all metrics, and considering the need to balance token usage and performance, these few-shots with extra examples are considered less effective.

Despite the improvements in query rewriting, initial experiments integrating this technique into the pipeline yielded results inferior to the improved Naive RAG, as shown below:

| Experiment                          | ROUGE-1 | BERT F1 | SemScore |
|-------------------------------------|---------|---------|----------|
| Naive RAG, simple baseline          | 0.307   | 0.699   | 0.871    |
| Best Improved Naive RAG             | 0.397   | 0.737   | 0.907    |
| Only Question Rewriting             | 0.353   | 0.716   | 0.899    |
| Question Rewriting + Original Query | 0.349   | 0.714   | 0.895    |

*Table 4.9: Performance comparison of different RAG implementations, focusing on the impact of adding question rewriting to the pipeline pre-retrieval process.*

The drop in performance suggests that query rewriting alone may introduce some additional noise, given that from the original query, in this case, five different questions are generated by GPT-4o trying to cover different aspects of the user request, and for each one of these questions the retrieval part of the RAG extracts the most relevant chunks. This process could increase the volume of retrieved chunks for the final generation, so it may also introduce noise, negatively affecting the final output. Additionally, as it can be seen from the results, including both the original and rewritten queries may have led to an increased retrieval of redundant or less informative chunks, reducing overall content effectiveness.

The context curation is a very important part of the system, and directly inputting the retrieved information into the LLM may not be ideal, as it can introduce excessive noise and redundant information, interfering with the model's generation. The context becomes too lengthy having a "Lost in the middle problem", that represents the fact that given too much information, the LLM tends to only focus on the beginning and the end of the long text, forgetting the middle [36]. Thus, given these results, further processing is applied to refine the retrieved content before passing it to the model.

The additional filtering consists of two steps. First, rewritten queries are filtered based on their similarity to the original query, preventing the propagation of erroneous or misleading reformulations. This step diminishes generation errors and hallucinations in the question rewriting. After different experimentations, the optimal similarity threshold was determined to be 0.3, yielding the best results while balancing both the need to filter out unwanted rewritten questions, and retaining those that provide different perspectives on the original question, thereby enhancing retrieval. Then, the second step to further process the retrieved context is given by the reranking of the extracted chunks to maintain the most relevant results first, acting both as an enhancer and as a filter [15].

The followed approach is known as RAG-Fusion, which combines RAG and Reciprocal Rank Fusion (RRF) by generating multiple queries, reranking them with reciprocal scores and fusing the documents and the scores. The RAG pipeline, using RAG-Fusion, has some extra steps. Once the original query is received, the pipeline passes it to GPT-4o model to generate five new search queries based on the original one, to obtain a better formulation, and the algorithm then performs vector search to find the most relevant chunks for each rewritten question. In the current implementation, the rewritten questions are also filtered in order to keep only the ones that actually make sense for the search; and instead of sending the relevant chunks together with the original query to the final LLM used for generation, RRF is applied first. RRF is an algorithm used in search to assign scores to every extracted chunk and order them according to the scores. The RRF scores are assigned, accumulating the score with previous scores for the same chunk, using an inverse-rank formula with a smoothing factor  $k$ , which balances the influence of highly ranked and lower-ranked chunks. Once all the scores are computed and accumulated, the chunks are passed, in this case alongside the original question, to the LLM for the output generation [37]. The following figure illustrates the RAG-Fusion process.

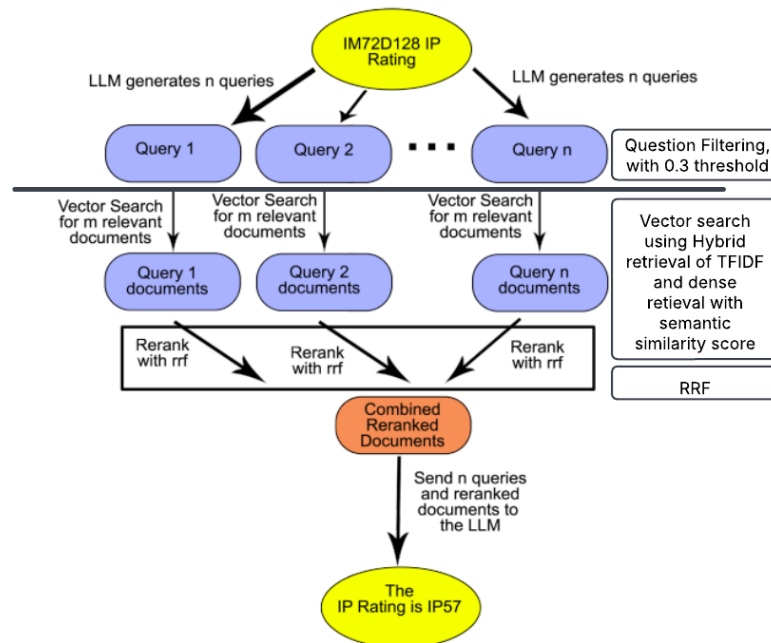


Figure 4.11: **RAG-Fusion** [37]: combines RAG and RRF to obtain a better context curation and reduce the 'lost in the middle' problem, giving first to the LLM the chunks with the highest RRF scores.

The experiments explored different filtering thresholds and RRF parameter values to optimize retrieval effectiveness. RRF helped mitigate the impact of suboptimal query variations by aggregating ranking signals from multiple retrieval outputs, effectively improving the diversity and relevance of retrieved results. In the RRF, the smoothing factor  $k$  is typically set to 60, given that this value is the one that from various empirical experiments yields the best results across various datasets and retrieval tasks; and this value provides a good balance between the influence of the best and the worst ranked elements, giving robust results. For this application, some experiments were also performed trying to tune  $k$ , but the system doesn't seem to benefit much from tuning this factor. Moreover, to correctly interpret the next results, it is important to highlight that, given the lower scores obtained integrating both the original and rewritten queries, all subsequent RRF experiments were conducted exclusively with the rewritten queries to reduce redundancy.

| Experiment                          | ROUGE-1 | BERT F1 | SemScore |
|-------------------------------------|---------|---------|----------|
| Naive RAG, simple baseline          | 0.307   | 0.699   | 0.871    |
| Best Improved Naive RAG             | 0.397   | 0.737   | 0.907    |
| Only Question Rewriting             | 0.353   | 0.716   | 0.899    |
| Question Rewriting + Original Query | 0.349   | 0.714   | 0.895    |
| RAG-Fusion ( $k=60$ )               | 0.367   | 0.722   | 0.896    |
| RAG-Fusion ( $k=30$ )               | 0.369   | 0.722   | 0.896    |
| RAG-Fusion ( $k=10$ )               | 0.366   | 0.721   | 0.895    |

**Table 4.10:** *Performance comparison of different advanced combinations of retrieval strategies, using question rewriting, a filtering threshold, and RRF.*

The best performing experiment for improving the Advanced RAG, which incorporates question rewriting, filtering, and RRF, is obtained using a similarity threshold for the questions equal to 0.3 and the RRF smoothing factor  $k = 30$ , achieving: ROUGE-1 of 0.369, BERT-F1 of 0.722, and SemScore of 0.896. However, given that the improvement over the implementation with  $k = 60$  is minimal and considering the proven experimental robustness of this  $k$ -value in other applications, the RAG-Fusion implementation with  $k = 60$  will be selected as the optimal configuration for the Advanced RAG pipeline.

The improvement obtained by adding the filtering and the RRF algorithm demonstrates how RAG-Fusion can mitigate the noise introduced by aggressive query rewriting, achieving a better retrieval and, consequently, an improved overall system with an improved quality generation. However, despite these enhancements, the Advanced RAG still does not surpass the improved Naive RAG results, highlighting the complexity of optimizing retrieval mechanisms. At the same time, the obtained scores underline the importance of a real-world validation, given that one possible explanation of the obtained results is that the synthetic evaluation set consists of well-structured queries, thus, the query rewriting

does not provide any benefit and instead introduces additional noise by increasing the number of extracted chunks given to the model. In contrast, the real world user queries are often badly written, ambiguous, and poorly structured, with misleading information, missing context and unclear phrasing. Thus, in the practical chatbot usage, query rewriting combined with filtering and RRF could prove crucial in enhancing robustness and improving the generation quality. Therefore, real-world validation is necessary to determine whether the observed performance degradation is an artifact of the evaluation set or a genuine limitation of the approach; and for this reason both best-performing configurations of the improved Naive RAG and the Advanced RAG will be deployed in production to perform A/B testing, comparing the two versions against each other to determine which one performs better in the real world practical application. This deployment will also facilitate the collection of human evaluations, which will be leveraged for future ad hoc improvements based on real user interactions.

#### 4.4.2. Comparison of LLM generation

The selection of the LLM together with the provided prompt is a fundamental aspect of the RAG implementation, affecting the final response of the system. Different LLMs were explored to generate the final response, evaluating their effectiveness on various configurations of the RAG pipeline. The best performing configuration was achieved using Gemini as LLM for the final generation, within the improved Naive RAG configuration, with the best hybrid retrieval explained in the previous sections. This combination consistently outperformed other setups in terms of lexical accuracy and semantic precision, making it the most effective choice for generation based on the evaluation set used for this study.

| Model             | Retrieval Type | ROUGE-1 | BERT-F1 | SemScore |
|-------------------|----------------|---------|---------|----------|
| Gemini            | TF-IDF Hybrid  | 0.397   | 0.737   | 0.907    |
| GPT-4o            | TF-IDF Hybrid  | 0.370   | 0.720   | 0.900    |
| Llama3.2          | TF-IDF Hybrid  | 0.368   | 0.728   | 0.910    |
| Llama3.2-instruct | TF-IDF Hybrid  | 0.352   | 0.720   | 0.902    |
| Gemini            | BM25 Hybrid    | 0.396   | 0.734   | 0.907    |
| GPT-4o            | BM25 Hybrid    | 0.371   | 0.724   | 0.900    |

Table 4.11: Performance comparison of different LLM models for the final output generation, with diverse retrieval implementations for the RAG pipeline.

In analyzing the obtained results, it is important to consider the effect of the prompt tuning. In this thesis, the prompting strategy was optimized specifically for Gemini, which may have influenced the evaluation in its favor. Since each LLM has a unique way to interpret the prompts, based on the specific training, using a prompt fine-tuned for a model

across all models may not fully leverage the strengths of the other LLMs, underperforming in the final scores. Observations from other studies and some experiments suggest that Gemini handles better the long, structured instructions, as the ones that are given in input in this case, while GPT-4o tends to prefer more direct and concise prompts. So, since the prompt optimized for Gemini was used across all models explored for the final generation, it is possible that GPT and Llama models could have achieved better scores if they were given a prompt tailored to their architecture. Therefore, while Gemini achieved the highest overall performance, part of this result may be attributed to prompt tuning and the iterative improvements of the pipeline tailored for this model, rather than inherent model superiority. Additionally, unlike Gemini, the achieved scores across different metrics suggest that the GPT-4o model may provide better results when using the hybrid retrieval with BM25, instead of TF-IDF. It is also important to consider, that the various parts of the pipeline were improved with the final generation using Gemini and that there are complex interconnections between retrieval and generation. The LLM used in the final step interacts differently with the retrieval strategy, and optimizations for one model may not necessarily generalize well to others.

Based on these considerations, the model selected for deployment is Gemini, but further prompt specific evaluations are necessary. Future research could focus on rerunning the tests with prompts tuned for other models assessing if their capabilities improve or if Gemini remains the best choice for the task at hand. Additionally, other variations of the pipeline discussed in this thesis could be tested with different models to assess whether the observed retrieval-generation interconnections are specific to the chosen configuration or if they hold more generally. Furthermore, some real-world evaluation could be gathered deploying two identical pipelines with only different final LLM for the generation, performing A/B testing with real user interactions and then iteratively improving the system based on user feedback. A more advanced improvement that can be implemented in future deployments of the system, but much more expensive, needing extensive experimentation, can rely on adaptive model selection, using different LLMs for different query types received from the user, deciding from additional experiments which model behaves best for each type of possible question, thus, the system may become much more flexible and capable of optimizing responses based on the specific characteristics of each interaction.

## 5 | Evaluation Metrics and Testing Framework

To assess the RAG-based chatbot performance on the customer support task of the ERP system, it was necessary to define a testing framework, which represents the combination of practices and tools to test the implementation efficiently. The evaluation process of the chatbot is focused on ensuring that the generated answers are aligned with the human expectations, but given the complexity of the task, traditional evaluation metrics and procedures were not applicable. The main elements to define, in order to evaluate the system in an automated and reliable way, are: the evaluation set of question-answers (QA) pairs and related extracted context, and the evaluation metrics that align more with human judgment. This framework leverages LLMs for both the evaluation set creation and the evaluation itself, and this chapter delves deeper into the methodology used to construct the synthetic evaluation set (given that a real one was not available, and that it was not possible to gather one during this implementation), and explains in detail the evaluation metrics used before, in this thesis, to measure the RAG-pipeline quality.

### 5.1. Testing Framework and Evaluation Set

The synthetic evaluation set was generated using a structured LLM-powered pipeline, with various steps, employing LLMs both for QA generation and as a judge for the quality of the generated pairs. The evaluation set creation starts from the augmented and preprocessed documents, ensuring the data is in the optimal format for extracting meaningful QAs, thus, it is crucial to use the elaborated documents in order to include the image description previously generated using Gemini. Starting from these documents, that contain all the customer support knowledge gathered for the application, the content is divided into chunks in order to facilitate the QA generation. The documents are split using the recursive chunking strategy, with a chunk size of 3000 and an overlap of 500, to preserve as much context as possible, allowing a broad coverage of different information, when passing the chunks to the generative model.

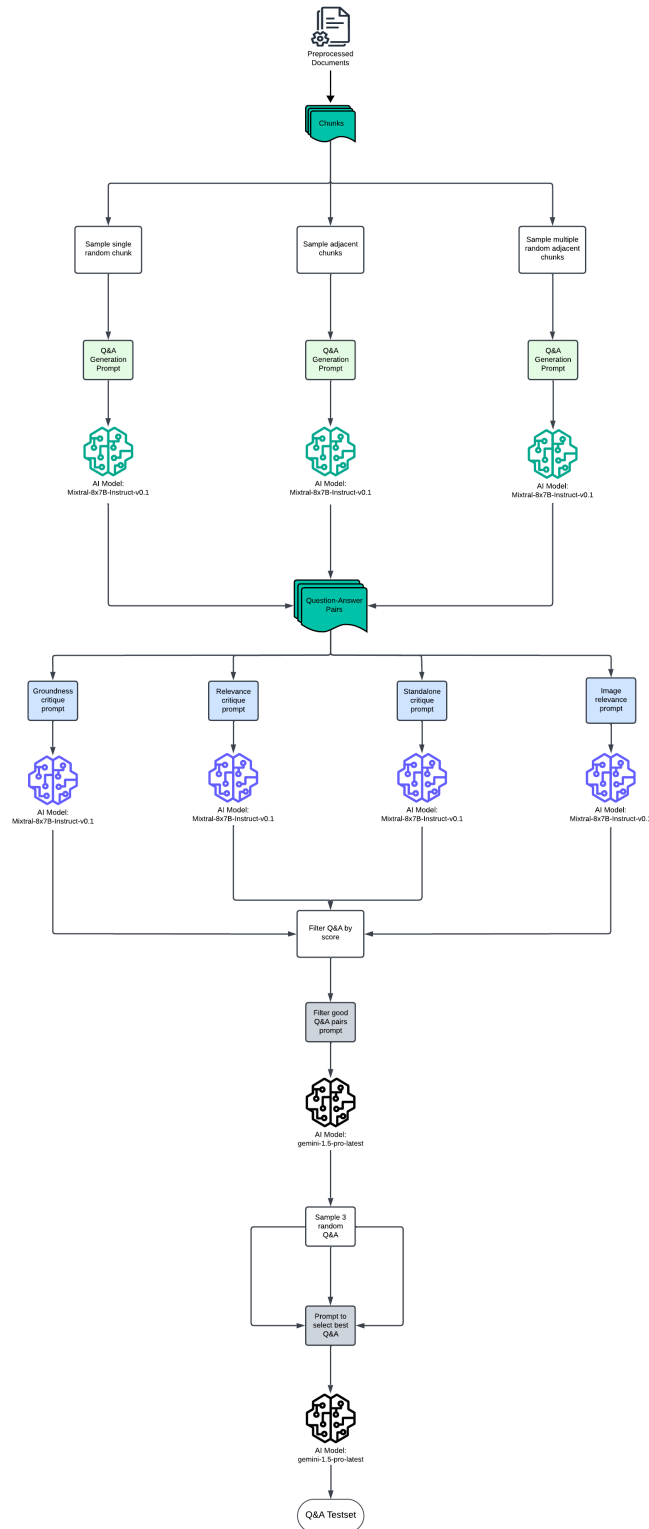


Figure 5.1: **Synthetic evaluation-set creation:** *the complete LLM-powered pipeline, employed to generate the best QA for the evaluation of the RAG-based chatbot, in the absence of a ground truth, to use for the system quality assessment.*

To generate QA pairs of different types and complexities, as illustrated in Figure 5.2, different sampling strategies were employed. These include: selecting a single random chunk, randomly sampling a chunk, extracting also its adjacent chunks within a certain range (with both preceding and following chunks of the randomly extracted one), and, finally, sampling random chunks with their adjacent chunks multiple times to combine more context. Balancing both randomness and coherence, and generating increasingly more complex questions, a certain amount of variability is ensured in the evaluation set.

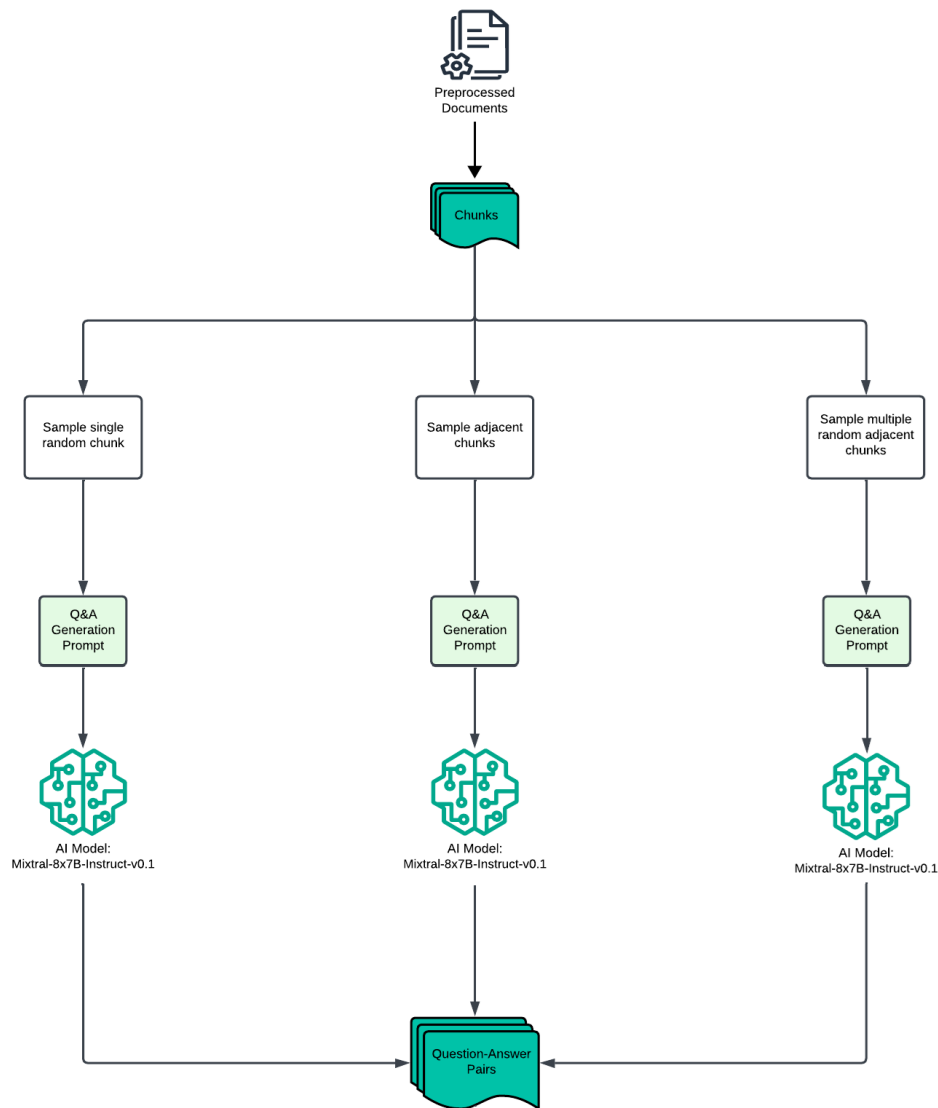


**Figure 5.2: Generate Question-Answers pairs using Mixtral-8x7B-Instruct-v0.1:** employing three different sampling strategies, random chunks from the preprocessed documents are provided to the Mixtral model with a specific prompt to generate human-like, useful QA pairs of increasing complexity.

The three sampling strategies used for chunk extraction were designed to extract only random chunks from the same document, preserving the context of the document, given that relationships between documents are not explicitly known. To explore the generation of more complex QA pairs, that require cross-document information, without using expert knowledge about the documents, which would require significant time and cost, an experiment was made to first cluster the documents before extracting the chunks. However, the resulting clusters were too loosely defined to meaningfully improve extraction, and the generated QAs were not considered useful for the application. While this approach did not yield positive results in this specific framework, it remains a potential sampling alternative, especially in domains that need complex cross-documents context queries. In this case, for now, the focus remains on refining the system to handle medium complexity queries, generated with the previously detailed sampling strategies. Over time, more sophisticated and real-world queries can be integrated, from actual human conversations with the implemented chatbot, gathering a much more reliable evaluation set.

After the sampling phase, each extracted chunk is given in input to the LLM along with a crafted prompt to generate the QA pairs. In this framework, the chosen LLM is *Mixtral-8x7B-Instruct-v0.1*, a powerful instruction-tuned model that can generate diverse QAs, starting from the provided context. Mixtral was selected primarily for its ability to handle a large context, up to 32k tokens, allowing randomly extracted chunks to be processed together with their adjacent context within the desired range; and it was also selected for its proficiency in Italian, since both the context and the generated QA pairs must be in Italian. This model is a sparse mixture-of-experts network with fast inference speed, and it is available as an open weight model with a permissive license on Hugging Face; and it demonstrates good performance across multiple benchmarks, ranking on the Hugging Face leaderboards [27]. Given all these considerations, and after conducting some experiments, also with other models, Mixtral was considered a balanced choice for generating the QAs pairs of the evaluation set, optimizing both cost and performance.

However, not all the generated QA pairs are of the same quality, thus, the pipeline doesn't stop here, but it is extended with rigorous filtering processes, leveraging LLM as a judge to ensure the highest quality possible for the evaluation dataset, in order to have a robust and reliable evaluation. To obtain the desired evaluation set, the generated QA pairs, as illustrated in **Figure 5.3**, are subjected to different critique agents, each one evaluating the pairs or only the questions based on specific criteria. The first criterion is the groundness, checking if the question can be uniquely and completely answered using only the given context, which is made up by the extracted chunks used to generate the pair. The second criterion is the relevance, which evaluates how useful and realistic the question

can be, specifically, whether a user might ask it when seeking for help from the help desk. Then, the third criterion regards the standalone clarity of the question, which measures how much the question makes sense on its own or whether additional context is necessary for the question to be comprehensible. For these three criteria, each critique agent has to make an evaluation on a single criterion and provide an overall rating on a scale from 1 to 5. Additionally, a fourth critique agent has to assess whether the question is related to entities such as images, screenshots, windows, and tables that appear in image descriptions, giving an output of 0 as the result if the question is image related, and '1' otherwise, allowing the subsequent filtering to remove all the questions related to images. In fact, after the critique agents provide their assessment, a filtering is applied. First, all the pairs related to images are removed, and then the average overall rating is computed from the first three agents, filtering out all the elements that do not score a rating of at least 4. Setting a high threshold ensures the creation of a high quality evaluation set; otherwise, the quality assessment of the RAG would not be reliable.

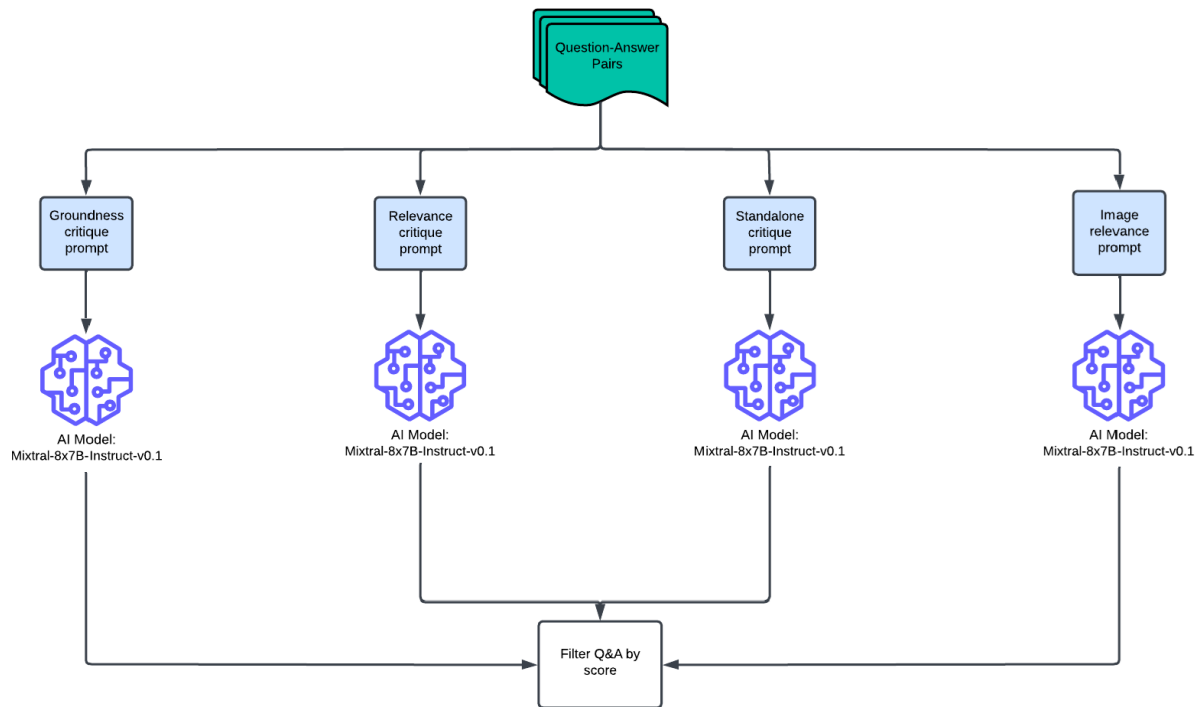


**Figure 5.3: Critique-Agents with Mixtral and QA pair filtering by score:** *using four critique-agents, the quality of the QAs is assessed; QAs related to image descriptions are filtered out, and only those that achieve a good score for all the agents, based on the criteria of groundedness, relevance, and standalone clarity are retained for the subsequent steps of the pipeline.*

This refinement step is not sufficient, given that the scoring is performed by an LLM, in this case Mixtral, used both for QA generation and as critique agent, therefore, errors and biases might persist. To avoid retaining poor QA pairs, some other filtering steps are implemented, creating a multistep refinement process to ensure high quality of the final evaluation set.

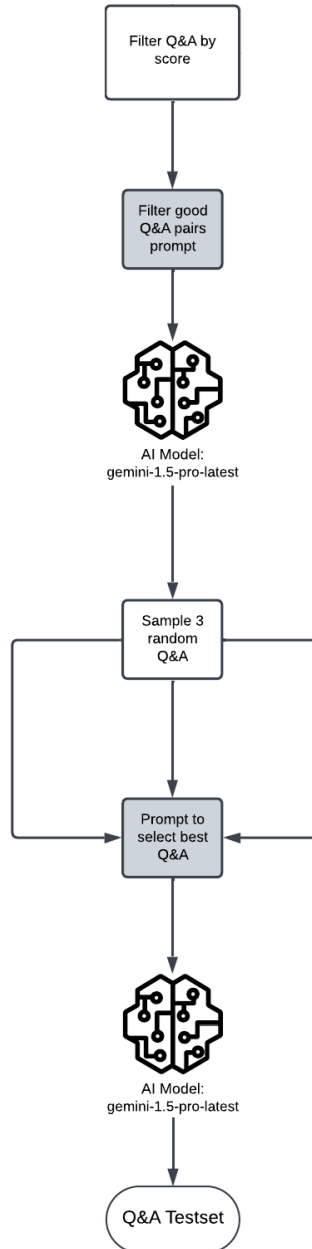


**Figure 5.4: Multistep final refinement with Gemini model as judge:** *additional quality control, using a different LLM as a judge, is necessary to reduce errors and biases from a single model, to balance perspectives, and obtain a high quality evaluation set.*

In the final steps of the pipeline there is a model shift from Mixtral to Gemini, specifically `gemini-1.5-pro-latest`, introducing an additional quality control layer, leveraging the strengths and the perspectives of a different model in order to reduce biases that may arise from using a single LLM. In this phase, Gemini first filters high quality QAs, evaluating each pair individually, specifically whether the QA pair can be relevant for an ERP system and whether the pair is clear enough, providing useful and practical guidance to the users. Then, to further enhance the control and balance LLM judgment with a degree of randomization, a second selection step is introduced, using again Gemini, this time to choose the best QA pair among three randomly extracted pairs, having not only an individual evaluation, but also a comparative one, introducing some variability in the filtering process. For a more in-depth analysis of the evaluation process and the prompts used for generation and judgment in this pipeline, refer to **Appendix B**, which provides the crafted instructions given to the LLM in the designed framework and further clarifies the multistep refinement process.

Despite the advantages of using this pipeline to generate an evaluation set, particularly in the absence of human experts or pre-existing datasets, there are still some challenges intrinsic to the idea of synthetically creating a dataset, given that it will never fully replicate the relevance and nuance of real human conversations, especially in complex and dynamic domains, like the ERP systems. Since this synthetic evaluation set is generated and judged using LLMs, it remains the risk of errors and biases, and model-specific tendencies, such as prioritizing fluency over accuracy and relevance. The multistep refinement process was specifically designed to mitigate these issues, requiring the QAs to pass through multiple filtering steps before acceptance, also changing the LLM used in the pipeline to increase robustness and reduce noise and biases in the final synthetic evaluation set.

## 5.2. Evaluation Metrics

Once the synthetic evaluation set is finalized, the second component of the evaluation framework consists of selecting the evaluation metrics, the ones that best align with human judgment. The RAG-based chatbot is used to generate responses for each question in the evaluation set, and these responses are compared against the synthetic reference answers, and three key metrics are employed in this case to evaluate the quality of the application and measure different aspects of textual similarity and relevance.

Since traditional metrics failed to provide meaningful insights and did not align with human assessment, after experimenting with different metrics, the ones selected for end-to-end evaluation of the RAG pipeline are: ROUGE-1, BERT-F1, and SemScore.

This section explains the selected metrics in detail (which were partially inspired by a paper on text generation evaluation [28]), in order to provide a deeper understanding of the results obtained in this thesis:

1. **ROUGE-1**: the Recall-Oriented Understudy for Gisting Evaluation metric is computed using uni-gram overlap between the words of the candidate answer and the reference. This is a measure of recall that captures the ability of the generated response to include important words from the reference, so it is based on words overlap, and not on the semantic meaning of the generation. It is considered a crucial metric for the task, given that precise terminology is required to guide the users in the complex ERP scenario, to ensure clarity when instructing the users to solve problems, and when explaining some concepts that need lexical accuracy;
2. **BERT-F1**: a BERT (Bidirectional Encoder Representations from Transformers) model is employed to measure the semantic similarity between the response and the reference answer, computing the F1 score, which is the harmonic mean of precision and recall, but unlike the traditional F1, it doesn't rely on exact token matching, focusing instead on meaning. This metric is used to complement the weaknesses of the ROUGE-1 metric, considering also the semantic correctness, ensuring that the chatbot's responses are relevant and aligned with the desired output, and not only lexically precise; and high BERT-F1 score indicates that the generated answers are close in meaning with the reference, even if the wording varies;
3. **SemScore**: similar to BERT-F1, SemScore also measures semantic similarity between the generated response and the reference. It measures the similarity using an embedding model and focusing on the semantic content of the model's output. SemScore captures nuanced relationships between words and phrases, ensuring that semantically equivalent answers receive high scores even if they use different wording and this makes SemScore especially valuable in evaluating chatbot-generated responses, where variation in phrasing is common but does not necessarily indicate a difference in meaning [29]. The SemScore proposed in the original paper [30], employed as embedding model the `sentence-transformers/all-mpnet-base-v2`, a Sentence-BERT, but after extensive experimentation and comparison with human evaluations, for this task `BAAI/bge-m3` was selected as embedding model. This model, in fact, reflected the human preferences and confirmed that responses from newer LLMs, such as `gpt-4o` are superior to those generated by older models, such as `gpt-3.5`, a behavior validated, also, by human experts, by comparing the responses on a batch of testing QAs.

Other traditional metrics, like BLEU, Diversity, Perplexity, and Readability, were also analyzed, but they were excluded from the final evaluation framework, given that they did not give actionable insights to improve the implementation. These metrics were explored in order to measure precision, repetitions, fluency, coherence, and user-friendliness in the responses, but given the high-quality generation of modern LLMs, these metrics had overall good results, but little variation across different implementations or did not correlate with human evaluation. For this reason, the three selected metrics are ROUGE-1, BERT-F1, and SemScore, which better aligned with human judgment, showing improvements with some implementations that actually gave responses that correlate with human preferences. This made these metrics valuable, guiding the choices of the implementation and the RAG-pipeline optimization.

For the intermediate phase of the pipeline, specifically for the retrieval step, traditional metrics such as hit rate, precision, recall, and NDCG, were used to evaluate the quality of the extracted chunks. However, accurately assessing retrieval performance proved challenging, as the referenced context was randomly sampled and varied significantly in size compared to the extracted chunks. This made it difficult to apply a semantic similarity score with a fixed threshold to compute these metrics reliably, so the retrieval metrics were used only to gain a general understanding of the improvements of the re-ranking and the hyperparameter tuning in the hybrid retrieval process, but then the results of the end-to-end evaluation were considered more influential and critical for assessing the pipeline enhancements. Therefore, this thesis places greater emphasis on evaluating the final generation quality, as it provides a more direct comparison with human judgment, while the complexity and variability of the extracted context made it difficult to align retrieval metrics with human assessment, reinforcing the decision to focus on the quality of the generated responses instead, considering the three end-to-end evaluation metrics detailed in this chapter to make important choices for the optimization of the chatbot.

### **Evaluation Metrics and Testing Framework conclusions.**

The implemented evaluation framework was inspired by a survey [31] on RAG evaluation, and by an open-source implementation, which demonstrates how a RAG system can be evaluated, by building a synthetic evaluation set and using LLM-as-a-judge to compute the accuracy of the system [32]. This thesis builds upon the ideas extracted from these sources, extending and adapting the framework experimentally for the task at hand, creating a multi-step refinement LLM-powered pipeline for the evaluation set generation and selecting the evaluation metrics that best align with human judgment, and providing actionable insights to optimize the RAG-based chatbot implementation.

### 5.2.1. Research questions on LLM juries

To explore the following research questions, a series of experiments were designed to evaluate the QA pairs generated by Mixtral with a panel of diverse models, from different families, including GPT, Llama, Gemma, Haiku, and other small models, using some simple and open-source versions of these models. The purpose of this diverse evaluator panel is to mitigate biases present in individual models and reduce dependence on a single perspective, having a more balanced and nuanced critique filtering the high-quality QAs, instead of using Mixtral to evaluate itself, which favors its outputs. The individual evaluations from these models are aggregated using two distinct methods: Majority Voting, where the most common decision is selected as the final judgment, and the Dawid-Skene method, a probabilistic approach that assigns different reliability weights to each judge based on inferred accuracy. The research is also extended to evaluate how LLM can be used as a judge or jury to evaluate the RAG pipeline end to end, with this additional automatic evaluation to make decisions upon the improvements that can be added to the implementation. In fact, given the cost of evaluating these models by human experts, many evaluations now rely on LLMs as judges to score the quality of outputs from other models; given that traditional metrics like BLUE and ROUGE can fail to analyze the intended property of interest when analyzing the generated text. These evaluations, though, usually use a single model, the largest and most universally capable one as an evaluator, but this is both slow and costly, limiting the applicability [39].

The main problem in the creation of the Panel of LLM Evaluators (PoLL) is the selection of the best models to include and the correct prompt to get the most from the models, and these choices depend on the task at hand. It is also very crucial to decide which is the best aggregation technique for the model predictions and the performed experiments are focused on the comparison between the Majority Voting (MV) and the Dawid-Skene (DS) aggregation alternatives. Thus, the addressed questions during these experiments regard the introduction of a panel of diverse evaluators performing LLMs as a jury instead of LLMs as a judge, and how these models can be aggregated to obtain the optimal evaluation: a cost-effective and robust evaluation.

In the performed experiments the model has to choose which of two outputs is better, given the generated outputs using two different models. The research was extended using not only the proprietary data but also some conversations from Chatbot Arena, which is one of the most popular benchmarks to evaluate LLM end-to-end performance, and the Chatbot Arena crowdsourced annotations are used as ground truth for calculating the correlation between the evaluator models and human judgments, a ground truth missing

in the task of the company proprietary data, thus it was necessary to test on a new dataset to get more insights from these experiments.

**Can a diverse Panel of LLM Evaluators (PoLL)—a jury of heterogeneous models—produce more reliable, cost-effective, and less biased evaluations compared to relying on a single large model?**

A key motivation for replacing a single large judge with a panel of heterogeneous models is to reduce bias in evaluation. In this thesis in particular, in the evaluation framework, given that, at the first step, the Mixtral model is used first as different critique agents, and then as an evaluator of the overall quality of the question-answer pairs, which were though generated also by the same Mixtral model, there can be some intra-model bias, considering some QA good, even if they are not. This idea follows other studies [38] which highlight how it has become popular to use the latest best-performing model on a task both as generator and evaluator, but it was recognized that this can be not only costly but also introduce biases, given that the analyses pinpointed that the models often recognize and prefer their outputs over the ones of other models, so they may not be good critiques of themselves. Thus, the main intuition was to upgrade from single judges to panels to surpass the limitations identified by using a single LLM as an evaluator [39]. By adding a Panel of LLMs (PoLL) of different types, the idea is to improve the evaluation, reducing the bias to have a more robust evaluation step, relying on a jury instead of a single judge. Also, based on the considered judges, the PoLL can reduce the cost of the evaluation, and in general, running a collection of smaller models in parallel is faster than a single big model. Other studies [[39], [40], [41]] found out that using PoLL composed of many small models outperforms a single large judge, having less intra-model bias given the composition of disjoint model families and demonstrating higher correlation with human judgment while reducing cost and latency in the evaluation.

From the investigations, a Panel of LLMs (PoLL) was formed using small open-source versions of models from different families, including GPT, Haiku, Gemma, and Llama (e.g., "google/gemma-2-2b", "meta-llama/Llama-3.2-3B", "ComCom/gpt2-small", "fabianmmueller/deep-haiku-gpt-2"), along with other lower-performing models. The results indicated that incorporating models from diverse families helped reduce biases, as different models exhibited distinct errors and varying reliability depending on the task. However, due to the small size of these models and the complexity of the task, the overall performance of the panel was insufficient for use as a reliable evaluator. The evaluations on the Chatbot Arena dataset were nearly random, while some improvements were observed on the private company dataset when manually assessing the results. In these experiments,

a single standardized prompt was used, and different panel configurations were tested, ranging from 3 to 9 judges, but no substantial improvement was achieved. Thus, while, from human analysis, the diverse panel seems to help mitigate biases compared to using a single Mixtral model as an evaluator and offered cost reductions, it did not provide a significant advantage over Mixtral in terms of evaluation quality. The effectiveness of the panel varied depending on the composition of annotators and the specific task, making it challenging to achieve consistently strong results.

### **Does the Dawid-Skene (DS) aggregation method provide a meaningful improvement in evaluation reliability over simpler approaches like Majority Voting (MV)?**

The literature on crowdsourced label aggregation (originally by Dawid and Skene, 1979) suggests that estimating each annotator’s reliability can produce a more accurate “true” label than a simple vote. In this extended research direction, Dawid-Skene (DS) can be integrated with PoLL by: treating each LLM judge as an annotator and estimating a reliability/confusion matrix for each judge (through iterative Expectation–maximization), then computing a weighted aggregate (the DS estimate) that reflects the “latent” true judgment.

In assessing the reliability of MV and DS, findings indicate that both methods yield similar exact-match accuracy. Cohen’s kappa scores, which measure inter-annotator agreement, were close to zero or slightly negative, suggesting that both approaches exhibit low agreement with human judgment. Pearson and Kendall-Tau correlations also indicated minimal association with the human-annotated ground truth, raising concerns about the models’ ability to consistently capture human-like evaluation criteria.

A key factor influencing DS performance is the availability of multiple judgments per data point. The DS method, originally developed for crowdsourced label aggregation (Dawid and Skene, 1979), benefits from having several annotators per instance (e.g., 5–10 evaluations per data point). It estimates each annotator’s confusion matrix and re-weights them. Given that the data has limited redundancy, and does not have enough “repeated” judgments per item, DS gains little advantage over MV. In the experiments, it also seems that the models all behave similarly, being relatively weak and almost random, so aggregating them there are not many benefits. DS can have an improvement over MV only if the models are of different capabilities, otherwise, if all are good the results won’t change using DS, and even more, if all models are almost random DS cannot deem a model more reliable than the other, so there are no real improvements in the final predictions.

### Future directions and conclusions on replacing judges with juries

Given the inconclusive advantage of DS over MV, further experimentation is necessary. Some further investigations are proposed here to continue this research and get conclusive results on this promising approach.

An idea is to use stronger models, aggregating larger models (Ensemble models as in LLMBLender: *Mistral-7B*, *Gemma-7B*, and *Llama-3-8B* [41]) which will improve the results for both MV and DS, having a higher agreement with human judgment. Using a diverse set of stronger models the hypothesis is that their errors will differ, the accuracy will be higher than the random results obtained so far, and aggregating these models can give an advantage to DS over MV, given that for the stronger and larger models we can have that some annotators are more reliable than others, thus DS can estimate this reliability assigning more weight to the more accurate models, obtaining better outcomes than MV which weights the same all the models. Choosing stronger and larger models can be helpful if the models are diverse enough and do not share the same biases, instead, if models mostly agree DS does not add much value over MV, as highlighted by the performed experiments.

Other future investigations can explore different prompting for each model, as in PromptBlender and LLMBLender [41], which may give better results by leveraging each model's unique strengths in a more precise way with tailored instructions. Combining prompt and model optimization could further enhance the evaluation accuracy of the panel, obtaining evaluations as good as the ones from much larger models, maintaining all the benefits of using diverse smaller models, the cost-effectiveness, and the bias reduction.

The obtained results on evaluating the proprietary dataset and the Chatbot Arena dataset with a jury are almost equal to a random evaluation, thus it is not possible to conclude something on the advantages of DS over MV, but this is believed to be also due to the complexity of this task and the dataset, especially considering the use of small models for the panel. Thus, extending the experiments to additional datasets beyond Chatbot Arena could help assess whether evaluation effectiveness varies based on dataset complexity or human annotation quality.

In conclusion, these findings indicate that while PoLL can mitigate bias and reduce costs, the benefit of DS remains inconclusive due to dataset constraints and model performance limitations. Future research will focus on incorporating stronger models, optimizing prompts, and expanding the dataset, to further refine the evaluation framework.



## 6 | Conclusions and Future Developments

The research presented in this thesis explored different RAG strategies for improving chatbot performance, focusing on enhancing the indexing, the retrieval, and the generation of the pipeline incrementally. The results of the experiments demonstrate the trade-offs between different techniques and highlight the challenges in optimizing the process for real-world complex applications, such as the customer support of an ERP system.

One of the key takeaways from this research is that in the RAG pipeline, the text preprocessing step is less crucial than the curation of the dataset and the data augmentation, which enhance much more the retrieval step, adding relevant information to the context, while advanced preprocessing techniques do not yield substantial improvements given the robustness of the final LLM used for generation, that handles minor noise in the input text, such as misspellings. These findings suggest that future improvements should focus on continuously integrating the ERP documentation used during retrieval by the RAG, ensuring that the chatbot can stay aligned with evolving system functionalities, dynamically adapting the responses based on data changes. Additionally, another enhancement could involve metadata filtering, which would differentiate the retrieval strategies based on the user's role, permissions, and specific customizations, as well as the ERP system version installed by the user. This kind of filtering would make it possible to provide more personalized and context-aware responses, ensuring users receive information that is relevant to their specific setup.

Another key finding is that while the Advanced RAG, which incorporates question rewriting, filtering, and RRF, achieved an improvement over standard RAG configurations, it did not outperform the improved Naive RAG on the synthetic evaluation set. This suggests that the benefits of query rewriting may be context-dependent: in scenarios where the queries are already well-formed, rewriting can introduce noise rather than improve retrieval. However, in real-world applications, where user queries are often ambiguous and incomplete, query rewriting, when combined with filtering and RRF, could enhance robustness and improve generation quality. To validate the real-world effectiveness of

these techniques, both the best-performing improved Naive RAG and the Advanced RAG will be deployed for A/B testing. This will provide valuable insights into how each configuration performs with actual user queries, while also offering human evaluations to use for continuous system improvements. This also suggests the importance of a real world evaluation set, instead of a synthetically generated one. However, a significant part of this thesis focuses on the generation of the synthetic evaluation, which was necessary for an incremental improvement approach, creating a testing framework that in most cases aligns with human judgment. However, the automated quantitative evaluation introduced in this research, despite carefully selecting the evaluation metrics for the application at hand, remains only a substitute for more expensive human assessment. In fact, while implementing and evaluating the system, it became evident more than once that the best evaluation is the one that relies on human feedback, both from end-users and ERP system experts familiar with the private company's documentation. These two human evaluations and insights should be integrated in the future in order to further improve the system. Thus, human feedback is crucial, and implementing a feedback loop mechanism, such as Reinforcement Learning From Human Feedback (RLHF) could provide a systematic way to integrate user interactions and preferences into the ongoing system improvements.

With respect to the RAG-pipeline explained so far, it is also important to mention two key additional developments that were required and completed in order to finalize the implementation and make it viable for deployment:

1. **Integration of chat memory to handle multi-turn conversations:** the system was enhanced with a chat memory mechanism, keeping in the prompt the summary of the last three messages of the chat, in order to avoid excessive context expansion while preserving key concepts across interactions, allowing the user to have multi-turn natural and coherent conversations with the final chatbot implementation;
2. **Development of a user-friendly interface:** a simple and intuitive interface was designed and implemented using `Streamlit`, in order to meet usability requirements and have the system ready for deployment, enabling the accessibility of the user to the chatbot.

The developed solution successfully reached the main requirements of the ERP chatbot assistant. First, the chatbot is capable of identifying user issues and retrieving relevant information to guide users in problem resolution or simply provide the required information. This requirement is the most crucial one, thus it has been the main focus of the implementation and the optimization explored in this thesis, aiming to achieve the reduction of ticket volume creation. Additionally, with the final integrations, the chatbot also

achieves the requirement of having a user-friendly interface, ensuring easy accessibility for the user; and by also incorporating multi-turn conversation handling, the system ensured the necessary contextual continuity, enhancing the overall user experience. Moreover, the obtained results and improvements highlight the importance of combining the RAG implementation with prompt engineering, achieving cost-effective results, while respecting the constraints. This approach is a significantly less expensive alternative compared to fine-tuning a model. However, for further system improvements, the RAG implementation can be combined with extensive fine-tuning and continuous pretraining of both embedding models and LLMs used for generation.

Despite the achievements, some functionalities remain to be implemented in order to achieve all the requirements of the initial analysis. The main missing feature is the integration of the task-oriented chatbot, which has to, not only answer to user questions, but when necessary, to also guide the user through ticket creation. This enhancement would enable the system to manage the ticket creation workflow, minimizing human errors and facilitating help-desk operations. Furthermore, another important development consists of creating a built-in feedback mechanism, allowing the user to rate responses, suggest corrections, and leave insights in a structured format, in order to collect this feedback and use it for further improvements. Moreover, another possible refinement of the chatbot responses could consist of integrating a post-processing step using LLM as a judge to improve the output readability and clarity, refining the answer and also filtering incorrect or incomplete answers.

The performed experiments reveal the potential of a RAG-based chatbot implementation for ERP customer support, which enables the use of a private knowledge base, that the model was not explicitly trained on. The implementation highlights the importance of retrieval quality and curated datasets, but also the need for real-world evaluation, much more effective to optimize the system than the evaluation on synthetic data. The implementation deploys two chatbot configurations of different complexity levels enabling A/B testing, which allows for a data-driven, human-in-the-loop approach for future system refinements, optimizing the model based on human assessment. In the future, the goal of this project is to transition from a knowledge-based assistant to a more interactive, task-oriented chatbot. This evolution would make it even a more essential component for improving customer support, enhancing user satisfaction and the overall ERP experience.

This work provides a strong foundation for further research in ERP-focused AI assistants, establishing the basis for future customizations and enhanced implementations.



# Bibliography

- [1] Oracle. *What is ERP?* <https://www.oracle.com/erp/what-is-erp/>.
- [2] Wikipedia. *Enterprise Resource Planning*. [https://en.wikipedia.org/wiki/Enterprise\\_resource\\_planning](https://en.wikipedia.org/wiki/Enterprise_resource_planning).
- [3] John Smith and Jane Doe. *AI-based Conversational Agents: A Scoping Review from Technologies to Future Directions*. ResearchGate, 2023. <https://www.researchgate.net/publication/362899303>.
- [4] DeepAI. *Conversational Agent*. <https://deepai.org/machine-learning-glossary-and-terms/conversational-agent>.
- [5] Nidhi Lohia. *Customer Service Automation: Pros, Cons and Best Practices*. <https://hiverhq.com/blog/customer-service-automation>.
- [6] Cole Stryker and Jim Holdsworth. *What is NLP?* <https://www.ibm.com/topics/natural-language-processing>.
- [7] DeepLearning.AI. *Natural Language Processing*. <https://www.deeplearning.ai/resources/natural-language-processing/>.
- [8] Daniel Jurafsky and James H. Martin. *Speech and Language Processing, Third Edition*. Stanford University, 2024.
- [9] Unstructured. *Unstructured Open Source*. <https://docs.unstructured.io/open-source/introduction/overview>.
- [10] Tahir Saeed. *Chunking and Embedding Strategies in RAG*. <https://medium.com/tahir.saeed/chunking-and-embedding-strategies-in-rag>.
- [11] Greg Kamradt. *5 Levels of Text Splitting*. [https://github.com/FullStackRetrieval-com/RetrievalTutorials/blob/main/tutorials/LevelsOfTextSplitting/5\\_Levels\\_Of\\_Text\\_Splitting.ipynb](https://github.com/FullStackRetrieval-com/RetrievalTutorials/blob/main/tutorials/LevelsOfTextSplitting/5_Levels_Of_Text_Splitting.ipynb).
- [12] Wikipedia. *Machine Learning*. [https://en.wikipedia.org/wiki/Machine\\_learning](https://en.wikipedia.org/wiki/Machine_learning).

- [13] Wikipedia. *Neural Networks*. [https://en.wikipedia.org/wiki/Neural\\_network\\_\(machine\\_learning\)](https://en.wikipedia.org/wiki/Neural_network_(machine_learning)).
- [14] Wikipedia. *Large Language Model*. [https://en.wikipedia.org/wiki/Large\\_language\\_model](https://en.wikipedia.org/wiki/Large_language_model).
- [15] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang and Haofen Wang. *Retrieval-Augmented Generation for Large Language Models: A Survey*. - Shanghai Research Institute for Intelligent Autonomous Systems, Tongji University, Shanghai Key Laboratory of Data Science, School of Computer Science, Fudan University and College of Design and Innovation, Tongji University, 2024.
- [16] Databricks. *Introduction to RAG in AI development*. <https://docs.databricks.com/en/generative-ai/tutorials/ai-cookbook/rag-overview.html>.
- [17] Langchain-AI. *RAG from scratch*. <https://github.com/langchain-ai/rag-from-scratch?tab=readme-ov-file>.
- [18] Databricks. *RAG chain for inference*. <https://docs.databricks.com/en/generative-ai/tutorials/ai-cookbook/fundamentals-inference-chain-rag.html>.
- [19] Databricks. *Improve RAG data pipeline quality*. <https://docs.databricks.com/en/generative-ai/tutorials/ai-cookbook/quality-data-pipeline-rag.htm>.
- [20] Jianlv Chen, Shitao Xiao, Peitian Zhang, Kun Luo, Defu Lian, Zheng Liu. *M3-Embedding: Multi-Linguality, Multi-Functionality, Multi-Granularity Text Embeddings Through Self-Knowledge Distillation*. - BAAI, University of Science and Technology of China, 28 Jun 2024.
- [21] Restack. *Langchain Retriever Similarity Score*. <https://www.restack.io/docs/langchain-knowledge-retriever-similarity-score-cat-ai>.
- [22] Renyi Qu, Forrest Bao, Ruixuan Tu. *Is Semantic Chunking Worth the Computational Cost?* - Vectara Inc. and University of Wisconsin-Madison, 16 Oct 2024.
- [23] Mehdi Allahyari, Angelina Yang. *RAG - Vector Databases and Their Role in Enhancing Retrieval*. [https://mallahyari.github.io/rag-ebook/02\\_rag.html](https://mallahyari.github.io/rag-ebook/02_rag.html).
- [24] LangChain. *Facebook AI Similarity Search (FAISS)*. <https://python.langchain.com/docs/integrations/vectorstores/faiss/>

- [25] LangChain. *Vector stores*. <https://python.langchain.com/docs/concepts/vectorstores/>.
- [26] LangChain. *Retrievers*. <https://js.langchain.com/docs/concepts/retrievers>.
- [27] Mixtral AI team. *Mixtral of experts: a high quality sparse mixture-of-experts*. <https://mistral.ai/news/mixtral-of-experts/>.
- [28] Asli Celikyilmaz, Elizabeth Clark, Jianfeng Gao. *Evaluation of Text Generation: A Survey*. - Facebook AI and Microsoft Research Department, University of Washington, 18 May 2021.
- [29] Hugging Face Community Article by g-ronimo. *SemScore: Evaluating LLMs with Semantic Similarity*. <https://huggingface.co/blog/g-ronimo/semscore>.
- [30] Ansar Aynetdinov, Alan Akbik. *SemScore: Automated Evaluation of Instruction-Tuned LLMs based on Semantic Textual Similarity*. - Humboldt-Universität zu Berlin, 5 Feb 2024.
- [31] Hao Yu, Aoran Gan, Kai Zhang, Shiwei Tong, Qi Liu, Zhaofeng Liu. *Evaluation of Retrieval-Augmented Generation: A Survey*. -Tencent Company, McGill University, State Key Laboratory of Cognitive Intelligence, University of Science and Technology of China, 3 Jul 2024.
- [32] Aymeric Roucher. *Open-Source AI Cookbook documentation - RAG evaluation*. [https://huggingface.co/learn/cookbook/rag\\_evaluation](https://huggingface.co/learn/cookbook/rag_evaluation).
- [33] Microsoft Azure AI Developers: John Alexander, Kristine Toliver, Jill Grant, McLean Schofield. *Build advanced retrieval-augmented generation systems*. <https://learn.microsoft.com/en-us/azure/developer/ai/advanced-retrieval-augmented-generation>.
- [34] Xinbei Ma, Yeyun Gong, Pengcheng He, Hai Zhao, Nan Duan. *Query Rewriting for Retrieval-Augmented Large Language Models*. - Department of Computer Science and Engineering and Key Laboratory of Shanghai Education Commission for Intelligent Interaction and Cognitive Engineering of Shanghai Jiao Tong University, Microsoft Research Asia Microsoft AI, 23 Oct 2023.
- [35] Sasha Aickin. *How many few shot examples should you use?* <https://www.getlibretto.com/blog/how-many-few-shot-examples-should-you-use>.
- [36] N. F. Liu, K. Lin, J. Hewitt, A. Paranjape, M. Bevilacqua, F. Petroni, and P. Liang.

- Lost in the middle: How language models use long contexts.* - Microsoft AI, 20 Nov 2023.
- [37] Zackary Rackauckas. *RAG-Fusion: A new take on Retrieval-Augmented-Generation.* - Infineon Technologies, 21 Feb 2024.
- [38] Arjun Panickssery, Samuel R. Bowman, Shi Feng *LLM Evaluators Recognize and Favor Their Own Generations.* - 15 Apr 2024.
- [39] Pat Verga, Sebastian Hofstatter, Sophia Althammer, Yixuan Su, Aleksandra Piktus, Arkady Arkhangorodsky, Minjie Xu, Naomi White, Patrick Lewis *Replacing Judges with Juries: Evaluating LLM Generations with a Panel of Diverse Models.* - Cohere, 1 May 2024.
- [40] Hao Li, Cor-Paul Bezemer, Ahmed E. Hassan *Software Engineering and Foundation Models: Insights from Industry Blogs Using a Jury of Foundation Models.* - Queen's University and University of Alberta, Canada, 6 Jan 2025.
- [41] Hossein A. Rahmani, Emine Yilmaz, Nick Craswell, Bhaskar Mitra *JudgeBlender: Ensembling Judgments for Automatic Relevance Assessment.* - University College London, The Alan Turing Institute in London, Amazon, Microsoft Seattle and Microsoft Research Montréal, 17 Dec 2024.

# A | Appendix A

This appendix was thought to be used as a reference for researchers and developers looking to implement a similar RAG-based chatbot system, thus, it provides the main tools employed throughout the project. The development of the chatbot leveraged multiple tools from the NLP and Generative AI ecosystem; and the following technologies were used for the final implementation:

1. **Python:** the primary programming language used for developing the chatbot application, integrating various NLP and AI components;
2. **Data science and Natural Language Preprocessing libraries:** classical data science libraries used in python such as `pandas` and `numpy`, together with other libraries used more specifically for NLP tasks, for text analysis and text preprocessing, such as `NLTK`, `spaCy`, `language_tool_python` and `spellchecker`, used in the simple and the advanced data preprocessing pipelines;
3. **Unstructured library and API:** used for efficient text parsing and image extraction in order to handle the documents ingestion with advanced and efficient strategies;
4. **LangChain:** a composable framework used to build applications powered by LLMs, useful for implementing various parts of the RAG-based chatbot, especially the indexing and the retrieval; it was used for document loading and accurate chunking with different strategies, as well as for the creation of the `FAISS` vectorstore and the simple and hybrid retrieval mechanisms;
5. **HuggingFace, Ollama and Flag Embedding:** these are tools that provide access to pre-trained transformer-based models, allowing to download and manage open-source AI, such as embedding models and LLMs for the chatbot generation;
6. **Pay-as-you-go APIs:** balancing cost-effective results the mainly used APIs are `OpenAI API` and `Google API`, in order to use `GPT` and `Gemini` models for different experiments and for the final implementation;
7. **Streamlit:** used for the chatbot interface creation.



## B | Appendix B

This appendix is going to detail all the prompting experiments performed in this thesis to improve the LLMs generations, in different circumstances: the image-to-text generation, the final generation and the question rewriting generation. All the following prompts were given in Italian to the LLMs, in order to maintain consistency with the context and the solely used language across the implementation. Nevertheless, in this section will be reported the Italian prompt together with its English translation, to have a continuity in the thesis and make everything clear and understandable.

### Prompting experiments - Prompting for image-to-text generation

In order to augment the original text a simple prompt was provided to a LLM in order to obtain the descriptions of the images extracted from the documentation, and integrate their description with the rest of the information.

---

*prompt0 = "Fornisci una descrizione dell'immagine fornita. Sii chiaro nella spiegazione dei vari campi della sezione principale della videata e i possibili utilizzi."*

***prompt0 = "Provide a description of the given image. Be clear in explaining the various fields of the main section of the screen and their possible uses."***

---

The prompts were experimented with different models, but heavily describing the context did not yield better results.

---

*prompt1 = "Analizza l'immagine fornita, che rappresenta una videata o un'immagine relativa ad una sezione del software gestionale. Descrivi chiaramente ciascun campo presente nella videata, con il nome del campo, il suo scopo funzionale, e i possibili utilizzi pratici, concentrandoti sulle informazioni utili ad un agente per dare assistenza al cliente e aiutarlo nell'utilizzo del software. La descrizione deve essere concisa e facilmente comprensibile. Assicurati di: Fornire una descrizione chiara di ciascun campo, ma evita dettagli specifici come esempi con dati sensibili (es. nomi utenti, date specifiche, o informazioni private); e*

*illustrare i possibili utilizzi di ogni campo per rispondere a domande comuni da parte degli utenti o per l'assistenza tecnica."*

*prompt1 = "Analyze the provided image, which represents a screen or an image related to a section of the enterprise management software. Clearly describe each field present in the screen, including the field name, its functional purpose, and possible practical uses, focusing on the information useful for an agent to assist the customer and help them use the software. The description must be concise and easily understandable. Make sure to: Provide a clear description of each field while avoiding specific details such as examples with sensitive data (e.g., user names, specific dates, or private information); and illustrate the possible uses of each field to address common user inquiries or technical support needs."*

---

In fact, given the security policies of models, such as GPT-4V and Gemini, these detailed prompts were not considered appropriate to achieve the task.

---

*prompt2 = "Fornisci una descrizione dell'immagine fornita, descrivi chiaramente i vari campi presenti nella videata, con il nome del campo, il suo scopo funzionale e i possibili utilizzi pratici, focalizzandoti su come ciascun campo possa aiutare un operatore a rispondere a domande generiche di assistenza o a navigare nell'interfaccia. La descrizione deve essere chiara e facilmente comprensibile. Assicurati di: fornire una spiegazione chiara di ciascun campo senza dettagli specifici come esempi con dati sensibili (es. nomi, date o altre informazioni private); e mostrare come ogni campo possa essere utile per rispondere a domande comuni o per fornire assistenza tecnica."*

*prompt2 = "Provide a description of the given image, clearly describing the various fields present in the screen, including the field name, its functional purpose, and possible practical uses, focusing on how each field can help an operator respond to general support inquiries or navigate the interface. The description must be clear and easily understandable. Make sure to: provide a clear explanation of each field without specific details such as examples with sensitive data (e.g., names, dates, or other private information); and demonstrate how each field can be useful in answering common questions or providing technical support."*

---

Trying to reduce the context given in input to the model some improvements are observed, obtaining more descriptions, but the model still doesn't answer properly many times considering the images as part of a proprietary software.

---

*prompt3 = "Fornisci una descrizione dell'immagine fornita, descrivi chiaramente i vari campi presenti nella videata, con il nome del campo, il suo scopo funzionale e i possibili utilizzi pratici, focalizzandoti su come ciascun campo possa aiutare un operatore a rispondere a domande generiche di assistenza o a navigare nell'interfaccia. La descrizione deve essere chiara e facilmente comprensibile. Assicurati di: fornire una spiegazione chiara di ciascun campo senza dettagli specifici (es. nomi, date o altre informazioni non rilevanti alle funzionalità della videata); e di mostrare come ogni campo possa essere utile per rispondere a domande comuni o per fornire assistenza tecnica."*

*prompt3 = "Provide a description of the given image, clearly describing the various fields present in the screen, including the field name, its functional purpose, and possible practical uses, focusing on how each field can help an operator respond to general support inquiries or navigate the interface. The description must be clear and easily understandable. Make sure to: provide a clear explanation of each field without specific details (e.g., names, dates, or other information not relevant to the functionality of the screen); and demonstrate how each field can be useful in answering common questions or providing technical support."*

---

Thus, the best prompt was considered the most simple one, that doesn't give any knowledge about the software context, and passes the security checks, obtaining accurate descriptions of the fields and the elements in the interfaces.

### Prompts used in the generation of the synthetic evaluation set

The refined prompt passed to the LLM to generate the QA pairs is the one reported here.

---

*QA\_generation\_prompt = "Il tuo compito è scrivere una domanda e una risposta in Italiano dato il contesto testuale della documentazione del software aziendale. La tua domanda deve essere rispondibile con un'informazione specifica dal contesto. Se nel contesto ci sono errori grammaticali o morfologici, correggili nell'output fornito. La tua domanda deve essere formulata nello stesso stile delle domande che gli utenti potrebbero porre ad un helpdesk che si occupa di assistenza clienti per un software aziendale. Nella generazione della domanda non devi fare riferimento direttamente alle descrizioni delle immagini e non devi utilizzare informazioni relative a schermate, finestre e tabelle di dati presenti nelle descrizioni delle immagini. La tua domanda NON deve menzionare frasi come "secondo il passaggio", "nel contesto", "nella schermata" o "perché l'immagine mostra". Non fare domande che chiedano quanti elementi sono visualizzati o che riguardano dati e valori presi da descrizioni che possono derivare da immagini. Evita di fare riferimento alle descrizioni delle immagini o di usare dettagli visivi nella generazione della domanda, ma utilizza queste informazioni*

*se necessario solo per la risposta, considerando che la risposta deve guidare l'utente nella risoluzione delle problematiche e dei quesiti posti nella domanda. Puoi formulare domande come: "Ho un errore", "Come posso risolvere il problema", o "Come posso sistemare questa situazione?". La risposta dovrà fare riferimento alle informazioni fornite nel contesto, citando correttamente flag, nomi o altre informazioni pertinenti, senza mai rivelare esempi o dati sensibili. Domanda e risposta devono essere generate in Italiano.*

*Fornisci la tua risposta come segue:*

*Output:::*

*Domanda: (la tua domanda)*

*Risposta: (la tua risposta alla domanda)*

*Ora ecco il contesto:*

*Contesto: {context}*

*Output:::"*

*QA\_generation\_prompt = "Your task is to write a question and an answer in Italian given the textual context of the corporate software documentation. Your question must be answerable with specific information from the context. If there are grammatical or morphological errors in the context, correct them in the provided output. Your question must be formulated in the same style as questions that users might ask a helpdesk providing customer support for corporate software. When generating the question, you must not directly refer to image descriptions or use information related to screenshots, windows, and table headers present in image descriptions. Your question MUST NOT mention phrases like 'according to the passage,' 'in the context,' 'in the screenshot,' or 'because the image shows.' Do not ask questions about how many elements are displayed or concerning data and values taken from descriptions that may come from images. Avoid referring to image descriptions or using visual details in generating the question, but use this information if necessary only for the answer, considering that the answer must guide the user in solving problems and addressing the issues raised in the question. You can formulate questions like: 'I have an error,' 'How can I solve the problem,' or 'How can I fix this situation?'. The answer should refer to the information provided in the context, correctly citing flags, names, or other relevant details, without ever revealing examples or sensitive data. The question and answer must be generated in Italian.*

*Provide your response as follows:*

*Output:::*

*Question: (your question)*

*Answer: (your answer to the question)*

*Now here is the context:*

*Context: {context}*

*Output:::"*

---

The following four prompts are used by the critique agents to evaluate the quality of the generated QA pairs. The first three agents assesses a specific aspect of the QA pair and assign a score between 1 and 5. First, QA pairs related to image descriptions (evaluated by the last critique agent) are removed; then, the remaining pairs must achieve an average score above 4 on the first three key aspects: groundedness, relevance, and standalone clarity. This high threshold is necessary to maintain evaluation set quality, guaranteeing a reliable RAG assessment.

---

*question\_groundedness\_critique\_prompt = "Sarà fornito un contesto e una domanda. Il tuo compito è di fornire una valutazione per indicare quanto bene si possa rispondere in modo univoco alla domanda data con il contesto fornito. Dai la tua risposta su una scala da 1 a 5, dove 1 significa che la domanda non è affatto rispondibile con il contesto, e 5 significa che la domanda è chiaramente e univocamente rispondibile con il contesto. Fornisci la tua risposta esattamente nel seguente formato:*

*Output:::*

*Valutazione totale: (il tuo punteggio, come numero tra 1 e 5)*

*Output:::*

*Ora ecco la domanda e il contesto.*

*Domanda: question*

*Contesto: context*

*Output:::"*

*question\_groundedness\_critique\_prompt = "A context and a question will be provided. Your task is to provide an assessment to indicate how well the given question can be uniquely answered with the provided context. Give your answer on a scale from 1 to 5, where 1 means that the question cannot be answered at all with the context, and 5 means that the question is clearly and uniquely answerable with the context. Provide your answer exactly in the following format:*

*Output:::*

*Overall rating: (your score, as a number between 1 and 5)*

*Output:::*

*Now here is the question and the context.*

*Question: question*

*Context: context*

*Output:::"*

---

*question\_relevance\_critique\_prompt = "Ti sarà fornita una domanda. Il tuo compito è di fornire una "valutazione totale" che rappresenti quanto utile possa essere questa domanda*

*per gli utenti che chiedono assistenza all'help desk riguardo a specifiche funzionalità del software gestionale e la relativa documentazione. Dai la tua risposta su una scala da 1 a 5, dove 1 significa che la domanda non è per nulla utile, e 5 significa che la domanda è estremamente utile. Fornisci la tua risposta esattamente nel seguente formato:*

*Output:::*

*Valutazione totale: (il tuo punteggio, come numero tra 1 e 5)*

*Output:::*

*Ora ecco la domanda.*

*Domanda: question*

*Output:::"*

*question\_relevance\_critique\_prompt = "You will be given a question. Your task is to provide an 'overall rating' that represents how useful this question might be for users seeking help from the help desk regarding specific functionalities of the management software and its documentation. Give your answer on a scale from 1 to 5, where 1 means that the question is not useful at all, and 5 means that the question is extremely useful. Provide your answer exactly in the following format:*

*Output:::*

*Overall rating: (your score, as a number between 1 and 5)*

*Output:::*

*Now here is the question.*

*Question: question*

*Output:::"*

*question\_standalone\_critique\_prompt = "Ti sarà fornita una domanda. Il tuo compito è di fornire una "valutazione totale" che rappresenti quanto questa domanda sia indipendente dal contesto. Dai la tua risposta su una scala da 1 a 5, dove 1 significa che la domanda dipende da informazioni aggiuntive per essere compresa, e 5 significa che la domanda ha senso da sola. Ad esempio, se la domanda si riferisce a un contesto particolare, come "nel contesto" o "nel documento", la valutazione deve essere 1. Le domande possono contenere termini tecnici o acronimi e ricevere comunque una valutazione di 5: deve semplicemente essere chiaro per un operatore con accesso alla documentazione di cosa tratta la domanda. Fornisci la tua risposta esattamente nel seguente formato:*

*Output:::*

*Valutazione totale: (il tuo punteggio, come numero tra 1 e 5)*

*Output:::*

*Ora ecco la domanda.*

*Domanda: question*

*Output:::"*

*question\_standalone\_critique\_prompt = "You will be given a question. Your task is to provide an 'overall rating' that represents how independent this question is from the context. Give your answer on a scale from 1 to 5, where 1 means that the question depends on additional information to be understood, and 5 means that the question makes sense on its own. For example, if the question refers to a particular context, such as 'in the context' or 'in the document,' the rating should be 1. Questions may contain technical terms or acronyms and still receive a rating of 5: it simply needs to be clear to an operator with access to the documentation what the question is about. Provide your answer exactly in the following format:*

*Output:::*

*Overall rating: (your score, as a number between 1 and 5)*

*Output:::*

*Now here is the question.*

*Question: question*

*Output:::"*

*image\_relevance\_critique\_prompt = "Ti sarà fornita una domanda. Il tuo compito è di valutare se la domanda riguarda entità come immagini, schermate, finestre o testate di tabelle che appaiono nelle descrizioni delle immagini. Inoltre, valuta se la domanda potrebbe contenere dati sensibili derivanti dalle descrizioni delle immagini, come informazioni su schermate, immagini o tabelle. Se la domanda è probabilmente derivata da un testo che descrive il contenuto di un'immagine, restituisci un punteggio di 0. Se la domanda non riguarda le immagini o non contiene dati sensibili, restituisci un punteggio di 1. Fornisci la tua risposta esattamente nel seguente formato:*

*Output:::*

*Valutazione totale: (il tuo punteggio, che può essere 0 o 1)*

*Output:::*

*Ora ecco la domanda.*

*Domanda: question*

*Output:::"*

*image\_relevance\_critique\_prompt = "You will be given a question. Your task is to assess whether the question pertains to entities such as images, screen-shots, windows, or table headers that appear in image descriptions. Additionally, evaluate whether the question might contain sensitive data derived from image descriptions, such as information about screens, images, or tables. If the question is likely derived from text describing the content of an image, return a score of 0. If the question does not concern images or does not contain sensitive data, return a score of 1. Provide your answer exactly in the following format:*

*Output:::*

*Overall rating: (your score, which can be 0 or 1)*

*Output:::*

*Now here is the question.*

*Question: question*

*Output:::"*

After the use of the four critique agents and the high threshold filtering, the QA pairs are refined once more, now switching from Mixtral to Gemini, which has to evaluate the relevance and usefulness of the pairs in order to use them for the chatbot evaluation.

*messages = [ SystemMessage(content="Sei un critico che valuta coppie di domande e risposte per una FAQ di un software gestionale. Le coppie di domande e risposte devono soddisfare i seguenti criteri per essere considerate utili per valutare un chatbot destinato al supporto clienti di un software gestionale:*

- 1. **\*\*Rilevanza\*\***: Devono affrontare temi rilevanti per gli utenti di un software gestionale.*
- 2. **\*\*Logicità e utilità\*\***: Devono essere logiche e utili per fornire informazioni chiare e pratiche agli utenti.*

*Valuta la seguente coppia e decidi se è utile per testare un chatbot per il supporto clienti.*

*Fornisci un feedback strutturato e dettagliato seguendo il formato specificato."),*

*HumanMessage(content=f"**\*\*Domanda:\*\*** row['question'], **\*\*Risposta:\*\*** row['answer']*

*Valuta questa coppia di domanda-risposta e fornisci un feedback nel seguente formato:*

*Sì/No*

*Spiegazione delle risposte:*

- **\*\*Motivazione del perché la coppia domanda-risposta è considerata utile o meno\*\****
- **\*\*Motivazione sulla rilevanza\*\***: (Breve spiegazione, se rilevante o non rilevante)*
- **\*\*Motivazione sulla logica\*\***: (Breve spiegazione, se logica e utile o meno)*

*Indica anche eventuali miglioramenti necessari nella domanda o nella risposta.")]*

*messages = [ SystemMessage(content="You are a critic evaluating pairs of questions and answers for a FAQ of enterprise management software. The question-answer pairs must meet the following criteria to be considered useful for evaluating a chatbot intended for customer support of enterprise management software:*

- 1. **\*\*Relevance\*\***: They must address topics relevant to users of enterprise management software.*
- 2. **\*\*Logic and usefulness\*\***: They must be logical and useful in providing clear and practical information to users.*

*Evaluate the following pair and decide if it is useful for testing a chatbot for customer support. Provide structured and detailed feedback following the specified format."),*

*HumanMessage(content=f"Question:\*\* row['question'], Answer:\*\* row['answer']*  
*Evaluate this question-answer pair and provide feedback in the following format:*  
*Yes/No*  
*Explanation of the response:*  
*- Reason why the question-answer pair is considered useful or not*  
*- Reasoning about relevance: (Brief explanation of whether it is relevant or not)*  
*- Reasoning about logic: (Brief explanation of whether it is logical and useful or not)*  
*Also indicate any necessary improvements in the question or the answer.")]*

---

The final selection of best QAs to use in the evaluation, is performed with the following prompt, choosing the highest quality QAs among 3 randomly extracted pairs.

---

*messages = [ SystemMessage(content="Ti fornisco tre coppie di domande e risposte relative al contesto di un software gestionale. Il tuo compito è selezionare la coppia migliore da utilizzare per valutare un chatbot basato su Retrieval-Augmented Generation (RAG). Considera i seguenti criteri per scegliere la coppia migliore:*

- 1. Autocontenute: La domanda e la risposta devono essere complete e non richiedere informazioni aggiuntive per essere comprese. Assicurati che il contesto sia sufficientemente chiaro per capire senza ambiguità di cosa si stia parlando.*
- 2. Chiarezza: La domanda deve essere formulata in modo chiaro, sensato, con una formulazione diretta ed essere realistica nel contesto del software gestionale. Non devono esserci termini vaghi o ambigui che possano confondere e deve essere plausibile che un utente ponga questa domanda.*
- 3. Pertinenza: La risposta deve essere direttamente correlata alla domanda e comprensibile per un utente che utilizza il software gestionale.*

*Escludi domande che riguardano visualizzazioni o operazioni generiche senza spiegazioni contestuali chiare. Inoltre, escludi domande troppo vaghe o che trattano aspetti di visualizzazione senza un contesto chiaro, come quelle che chiedono genericamente <quali informazioni sono visibili in una tabella> senza specificare il contesto o la funzione richiesta. Evita anche domande che chiedono come visualizzare qualcosa senza spiegare di quale dato o schermata si sta parlando.")*,

*HumanMessage(content=f"Le tre coppie di domande e risposte sono:*

- 1. Domanda: qas[0][0] - Risposta: qas[0][1]*
- 2. Domanda: qas[1][0] - Risposta: qas[1][1]*
- 3. Domanda: qas[2][0] - Risposta: qas[2][1]*

*Seleziona solo la domanda e la risposta migliori in base ai criteri sopra descritti, senza alcun altro commento o spiegazione.*

*Rispondi nel formato:*

*Domanda: [domanda migliore]*

*Risposta: [risposta migliore]")]*

*messages = [ SystemMessage(content="I will provide you with three pairs of questions and answers related to the context of enterprise management software. Your task is to select the best pair to use for evaluating a chatbot based on Retrieval-Augmented Generation (RAG). Consider the following criteria to choose the best pair:*

- 1. **\*\*Self-contained\*\***: The question and answer must be complete and should not require additional information to be understood. Ensure that the context is sufficiently clear to understand without ambiguity what is being discussed.*
- 2. **\*\*Clarity\*\***: The question must be clearly formulated, sensible, directly structured, and realistic within the context of the enterprise management software. There should be no vague or ambiguous terms that might cause confusion, and it must be plausible that a user would ask this question.*
- 3. **\*\*Relevance\*\***: The answer must be directly related to the question and comprehensible for a user using the enterprise management software.*

*Exclude questions that concern generic visualizations or operations without clear contextual explanations. Also, exclude questions that are too vague or that deal with visualization aspects without a clear context, such as those that generically ask <what information is visible in a table> without specifying the context or the required function. Additionally, avoid questions that ask how to display something without specifying which data or screen is being referred to.")*,

*HumanMessage(content=f"The three pairs of questions and answers are:*

- 1. Question: qas[0][0] - Answer: qas[0][1]*
- 2. Question: qas[1][0] - Answer: qas[1][1]*
- 3. Question: qas[2][0] - Answer: qas[2][1]*

*Select **\*\*only the best question and answer\*\*** based on the criteria described above, without any other comments or explanations.*

*Respond in the format:*

*Question: [best question]*

*Answer: [best answer"]]*

---

This final prompt was prepared ad hoc based on the remained QAs, after applying all the filters, trying to address the specific problems, identified with an accurate analysis, aiming to enhance groundedness, clarity and relevance.

### Prompts used for the question rewriting implementation in Advanced RAG.

The following prompts are used both in Italian and English with different models trying to assess the best combination for question rewriting, specifying the AI task as an assistant and clarifying the context of the ERP system and the main instructions to take into consideration in order to give the required output in the desired format.

---

*template\_italian = "Sei un assistente modello linguistico AI. Il tuo compito è generare cinque versioni diverse della domanda fornita dall'utente per recuperare documenti rilevanti da un database vettoriale. Il contesto riguarda manuali tecnici di software di Enterprise Resource Planning (ERP).*

*\*\*Istruzioni:\*\**

- 1. Genera domande riscritte che mantengano il significato originale, esplorando diverse formulazioni e angolazioni.*
  - 2. Ignora le domande che non sono pertinenti ai manuali ERP o agli argomenti di informatica.*
  - 3. Fornisci le domande alternative in un elenco puntato separato da nuove righe.*
  - 4. L'output deve contenere solo le domande riscritte, senza spiegazioni o commenti.*
- Domanda originale: {question}"*

*template\_english = "You are an AI language model assistant. Your task is to generate five different versions of the question provided by the user to retrieve relevant documents from a vector database. The context involves technical manuals for Enterprise Resource Planning (ERP) software.*

*\*\*Instructions:\*\**

- 1. Generate rewritten questions that retain the original meaning while exploring different phrasings and angles.*
  - 2. Ignore questions that are not relevant to ERP manuals or IT-related topics.*
  - 3. Provide the alternative questions in a bullet-point list, separated by new lines.*
  - 4. The output should contain only the rewritten questions, without explanations or comments.*
- Original question: {question}"*
- 

The main problem of this prompt is that the model doesn't really understand which questions are out of scope, rewriting also some questions which are not relevant for the domain. Therefore, the prompt is improved to better specify the assistant scope and highlights the fact that it doesn't have to give in output re-written versions of questions that do not regard the software documentation.

---

*template\_italian = "**Sei un modello linguistico AI, che svolge il ruolo di assistente clienti per il software Panthera.** Il tuo compito è generare cinque versioni diverse della domanda fornita dall'utente per recuperare documenti rilevanti da un database vettoriale. Il contesto riguarda manuali tecnici di software di Enterprise Resource Planning (ERP). **Non fornire la riscrittura della domanda se non è relativa al contesto.***

*\*Istruzioni:\*\**

- 1. Genera domande riscritte che mantengano il significato originale, esplorando diverse formulazioni e angolazioni.*

2. Ignora le domande che non sono pertinenti ai manuali ERP o agli argomenti di informatica.
  3. Fornisci le domande alternative in un elenco puntato separato da nuove righe.
  4. L'output deve contenere solo le domande riscritte, senza spiegazioni o commenti.
- Domanda originale: {question}"

*template\_english = "You are an AI language model acting as a customer support assistant for the Panthera software. Your task is to generate five different versions of the question provided by the user to retrieve relevant documents from a vector database. The context involves technical manuals for Enterprise Resource Planning (ERP) software. Do not rewrite the question if it is not related to the context.*

*\*\*Instructions:\*\**

1. Generate rewritten questions that retain the original meaning while exploring different phrasings and angles.
2. Ignore questions that are not relevant to ERP manuals or IT-related topics.
3. Provide the alternative questions in a bullet-point list, separated by new lines.
4. The output should contain only the rewritten questions, without explanations or comments.

*Original question: {question}"*

---

By analyzing the results obtained with the new prompt, another issue is identified, the fact that when re-writing the question the model mentions a lot of times the ERP system, being repetitive and directly referencing the context explained in the prompt, but given that this is not present in the documentation, it mainly confuses the retrieval, so the prompt is changed in order to avoid the use of so specific technical terms, using a more general terminology, but still specifying the context of the question, which is critical to input into the model. Additionally, enlarging the prompt and better specifying the task and the output format, enhances the results and the question rewriting is finally deemed satisfactory.

---

*template\_italian = "Sei un modello linguistico AI che svolge il ruolo di assistente clienti per il software Panthera. Il tuo compito è generare cinque versioni diverse della domanda fornita dall'utente per recuperare documenti rilevanti da un database vettoriale. **Il contesto riguarda manuali tecnici di software per la gestione aziendale.** Non fornire la riscrittura della domanda se non è relativa al contesto. **Se nessuna delle domande è rilevante per il contesto del software fornisci come output solamente il tag: "OUT-OF SCOPE".***

*\*\*Istruzioni:\*\**

1. Genera domande riscritte che mantengano il significato originale, esplorando diverse formulazioni e angolazioni.

2. Ignora le domande che non sono pertinenti ai manuali di software gestionale o agli argomenti di informatica.

3. Fornisci le domande alternative in un elenco puntato separato da nuove righe.

4. L'output deve contenere solo le domande riscritte, senza spiegazioni o commenti.

**Svolgi la task solo per le domande rilevanti al contesto del software. In questo caso, cerca di migliorare la formulazione originale esplorando diverse angolazioni che aiutino a comprendere meglio il problema o la richiesta, rendendo più chiare e leggibili le domande per un utente generico.**

Domanda originale: {question}"

*template\_english = "You are an AI language model acting as a customer support assistant for the Panthera software. Your task is to generate five different versions of the question provided by the user to retrieve relevant documents from a vector database. **The context involves technical manuals for business management software.** Do not rewrite the question if it is not related to the context. **If none of the questions are relevant to the software context, provide only the tag: 'OUT-OF SCOPE' as output.***

**\*\*Instructions:\*\***

1. Generate rewritten questions that retain the original meaning while exploring different phrasings and angles.

2. Ignore questions that are not relevant to business management software manuals or IT-related topics.

3. Provide the alternative questions in a bullet-point list, separated by new lines.

4. The output should contain only the rewritten questions, without explanations or comments.

**Perform the task only for questions relevant to the software context. In this case, try to improve the original formulation by exploring different angles that help better understand the problem or request, making the questions clearer and more readable for a general user.**

Original question: {question}"

---

Building on the zero-shot, the impact of different few-shot examples is analyzed. Given the results, and the fact that by adding more examples the performance does not improve consistently, the few-shot reported below, with two examples, a positive and a negative one, is deemed as the best balance between token usage and effectiveness.

---

*template\_italian = "Sei un modello linguistico AI che svolge il ruolo di assistente clienti per il software Panthera. Il tuo compito è generare cinque versioni diverse della domanda fornita dall'utente per recuperare documenti rilevanti da un database vettoriale. Il contesto riguarda manuali tecnici di software per la gestione aziendale. Non fornire la riscrittura della domanda se non è relativa al contesto. Se nessuna delle domande è rilevante per il contesto del software fornisci come output solamente il tag: "OUT-OF SCOPE".*

***\*\*Esempi:\*\****

1. *Domanda originale: "Come posso specificare il tipo costo da assegnare al nuovo ambiente di commessa?"*

***Risposte:***

- *Qual è il procedimento per definire il tipo di costo da applicare a un nuovo ambiente di commessa?*
- *Come posso configurare il tipo di costo per un nuovo ambiente di commessa nel sistema?*
- *Quali sono i passaggi per assegnare un tipo di costo a un ambiente di commessa appena creato?*
- *Dove posso trovare l'opzione per impostare il tipo di costo di un nuovo ambiente di commessa?*
- *Come posso scegliere il tipo di costo da associare a un ambiente di commessa nel mio sistema?*

2. *Domanda originale: "Come si fa la carbonara?"*

***Risposta: OUT-OF SCOPE***

***\*\*Istruzioni:\*\****

1. *Genera domande riscritte che mantengano il significato originale, esplorando diverse formulazioni e angolazioni.*
  2. *Ignora le domande che non sono pertinenti ai manuali di software gestionale o agli argomenti di informatica.*
  3. *Fornisci le domande alternative in un elenco puntato separato da nuove righe.*
  4. *L'output deve contenere solo le domande riscritte, senza spiegazioni o commenti.*
- Svolgi la task solo per le domande rilevanti al contesto del software. In questo caso, cerca di migliorare la formulazione originale esplorando diverse angolazioni che aiutino a comprendere meglio il problema o la richiesta, rendendo più chiare e leggibili le domande per un utente generico. Domanda originale: {question}"*

*template\_english = "You are an AI language model acting as a customer support assistant for the Panthera software. Your task is to generate five different versions of the question provided by the user to retrieve relevant documents from a vector database. The context involves technical manuals for business management software. Do not rewrite the question if it is not related to the context. If none of the questions are relevant to the software context, provide only the tag: OUT-OF SCOPE."*

***\*\*Examples:\*\****

1. *Original question: "How can I specify the cost type to assign to the new job environment?"*

***Responses:***

- *What is the process for defining the cost type to apply to a new job environment?*
- *How can I configure the cost type for a new job environment in the system?*
- *What are the steps to assign a cost type to a newly created job environment?*
- *Where can I find the option to set the cost type for a new job environment?*
- *How can I choose the cost type to associate with a job environment in my*

*system?*

*2. Original question: "How do you make carbonara?"*

*Response: OUT-OF SCOPE*

*\*\*Instructions:\*\**

- 1. Generate rewritten questions that retain the original meaning while exploring different phrasings and angles.*
- 2. Ignore questions that are not relevant to business management software manuals or IT-related topics.*
- 3. Provide the alternative questions in a bullet-point list, separated by new lines.*
- 4. The output should contain only the rewritten questions, without explanations or comments.*

*Perform the task only for questions relevant to the software context. In this case, try to improve the original formulation by exploring different angles that help better understand the problem or request, making the questions clearer and more readable for a general user.*

*Original question: {question}"*

---

This prompt, given in Italian, combined with GPT-4o achieves the best results in question rewriting for the task at hand. In the following section will be, also, provided some graphs to better visualize the performance of different LLMs for the question rewriting, comparing the results both for Italian and English prompting.

Comparison of Metrics across different models with English prompt.

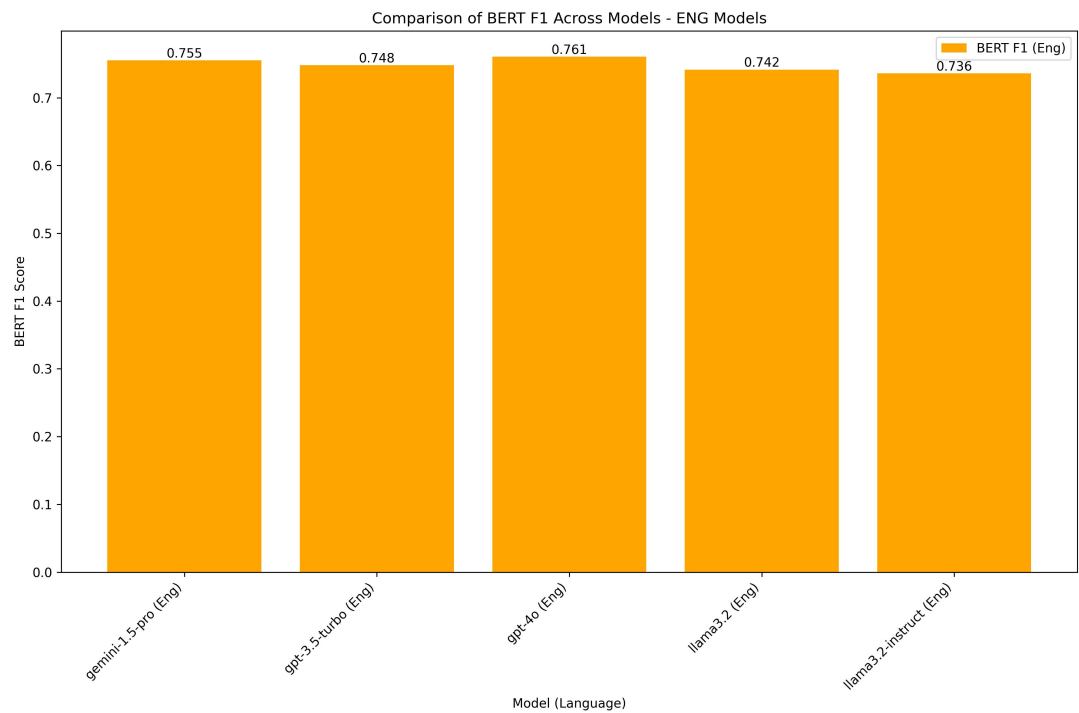


Figure B.1: *BERT F1 Score - English prompt.*

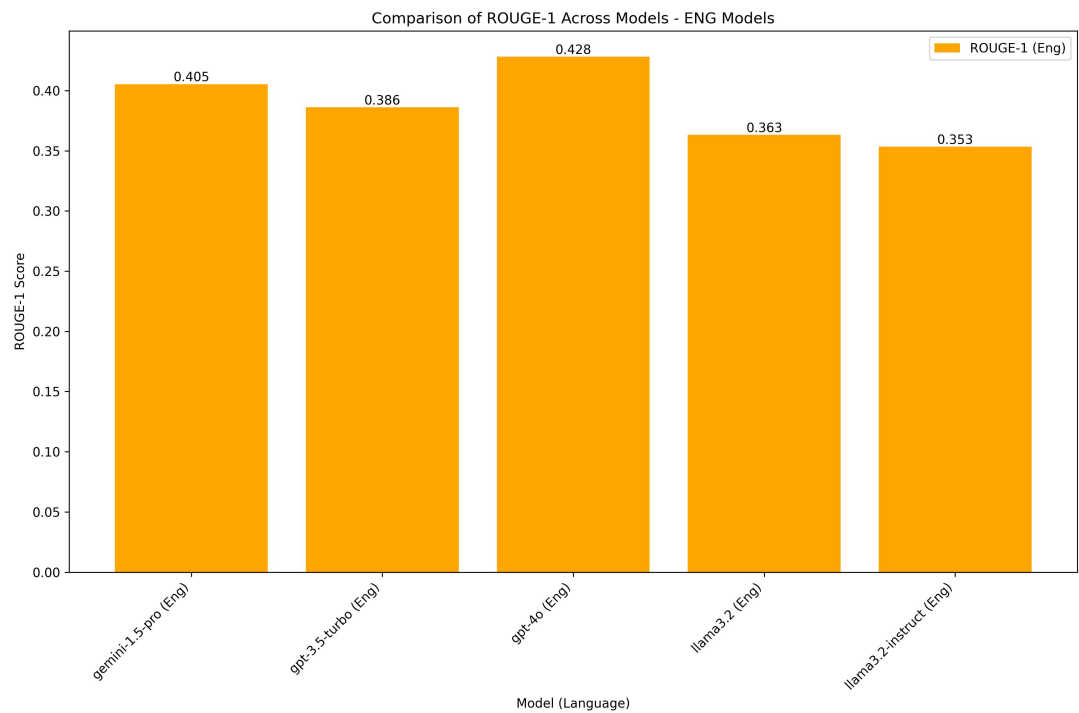


Figure B.2: *ROUGE-1 Score - English prompt.*

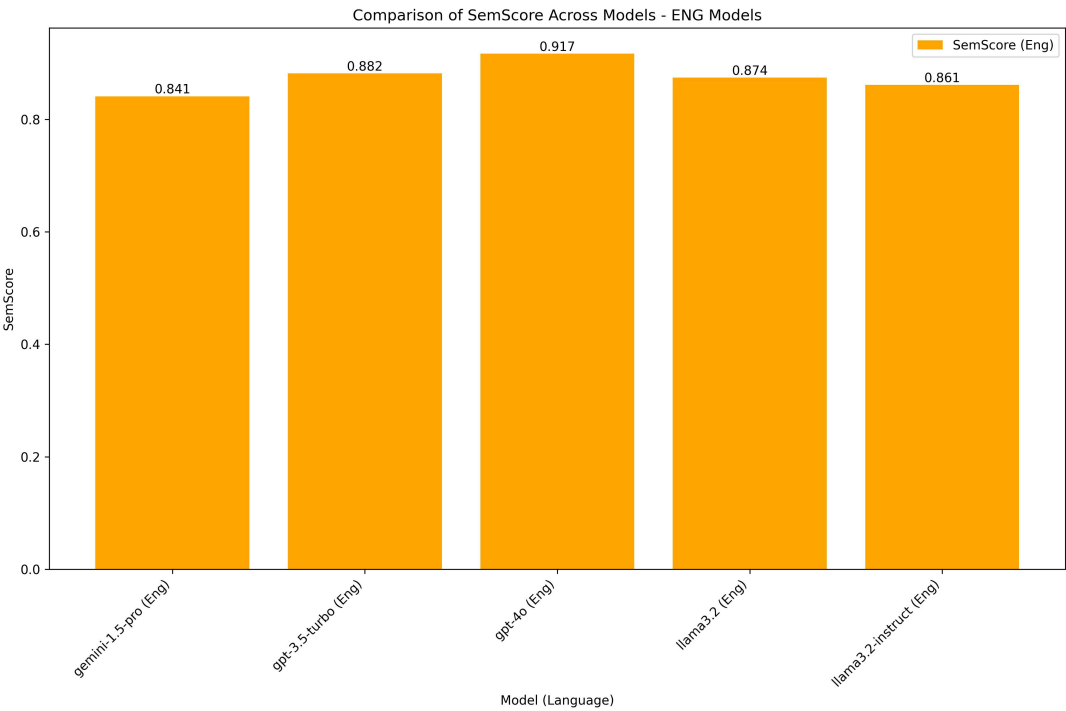


Figure B.3: *SemScore - English prompt.*

Comparison of Metrics across different models with Italian prompt.

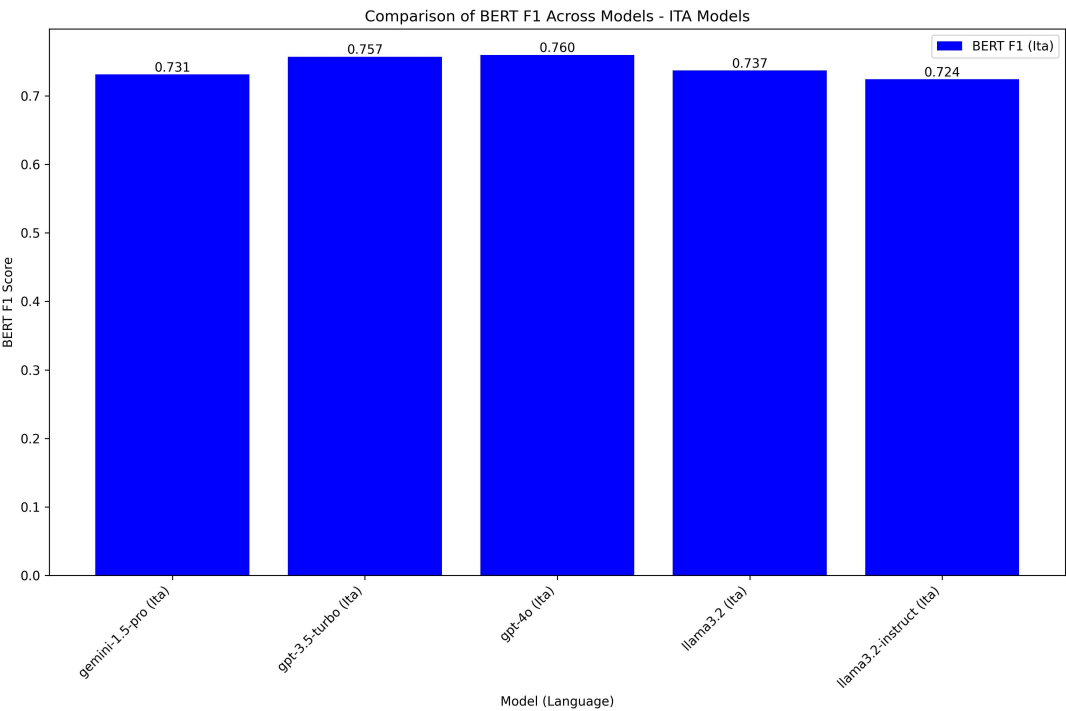


Figure B.4: *BERT F1 Score - Italian prompt.*

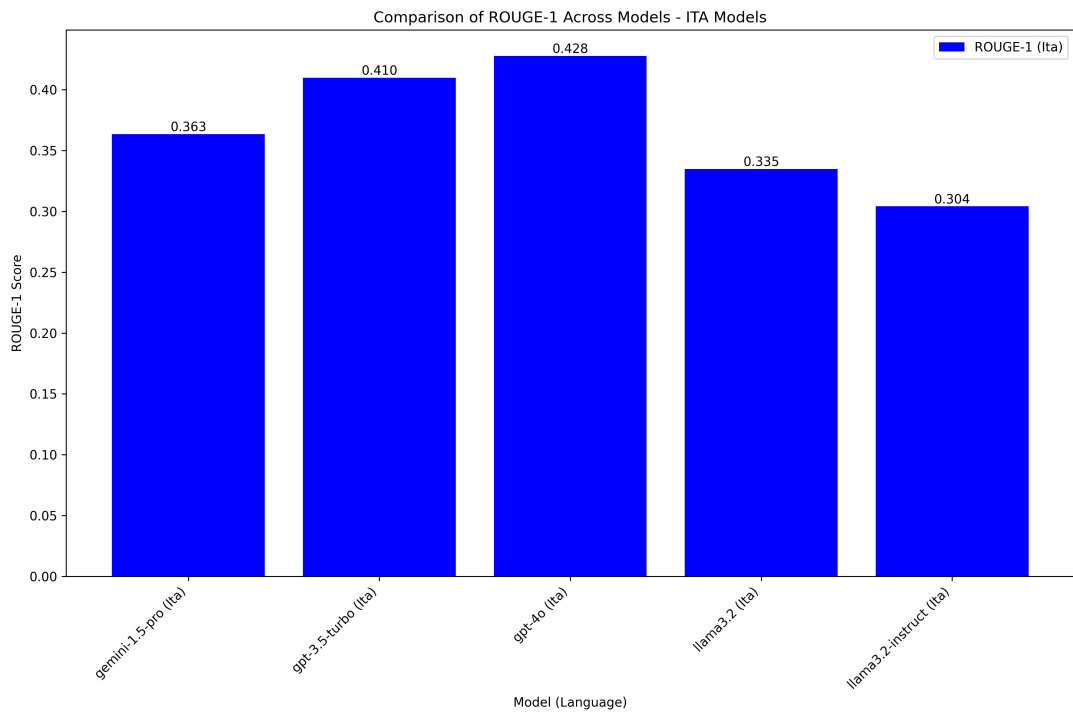


Figure B.5: *ROUGE-1 Score - Italian prompt.*

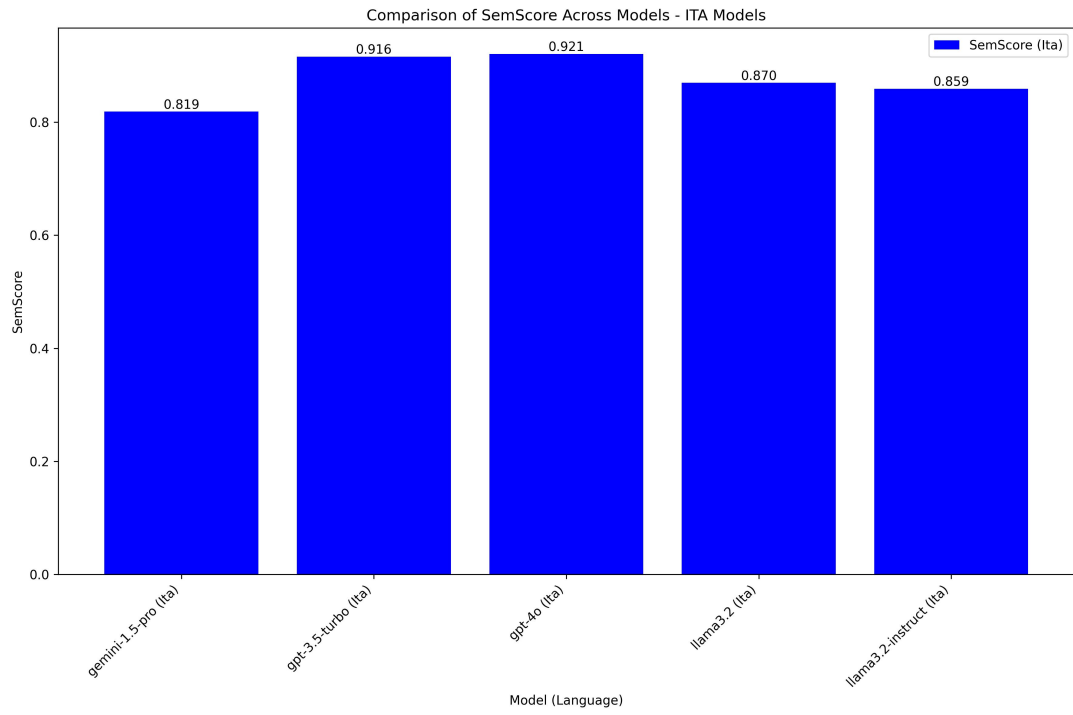


Figure B.6: *SemScore - Italian prompt.*

General Comparison across all models and metrics for both English and Italian prompts.

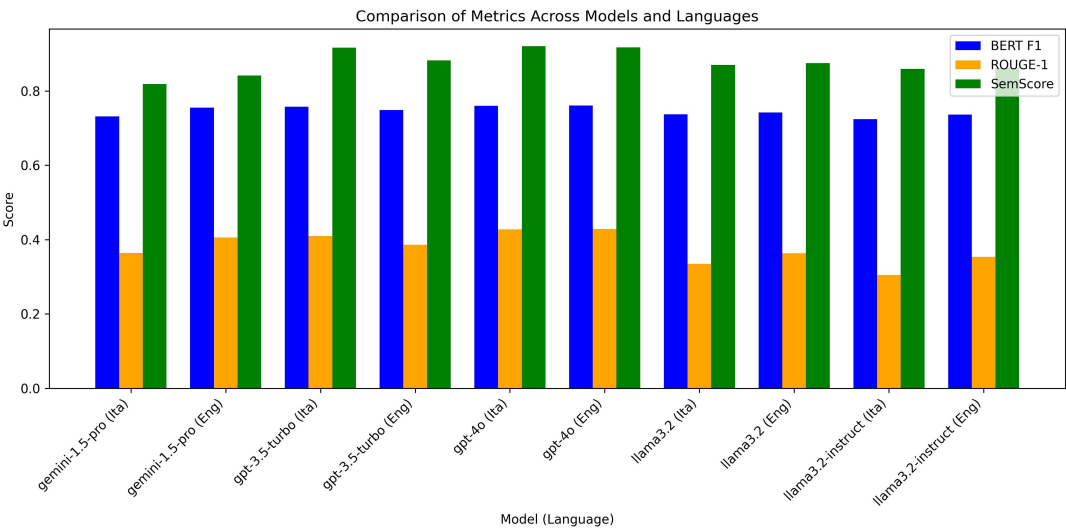


Figure B.7: Comparison of all metrics across different models and languages.

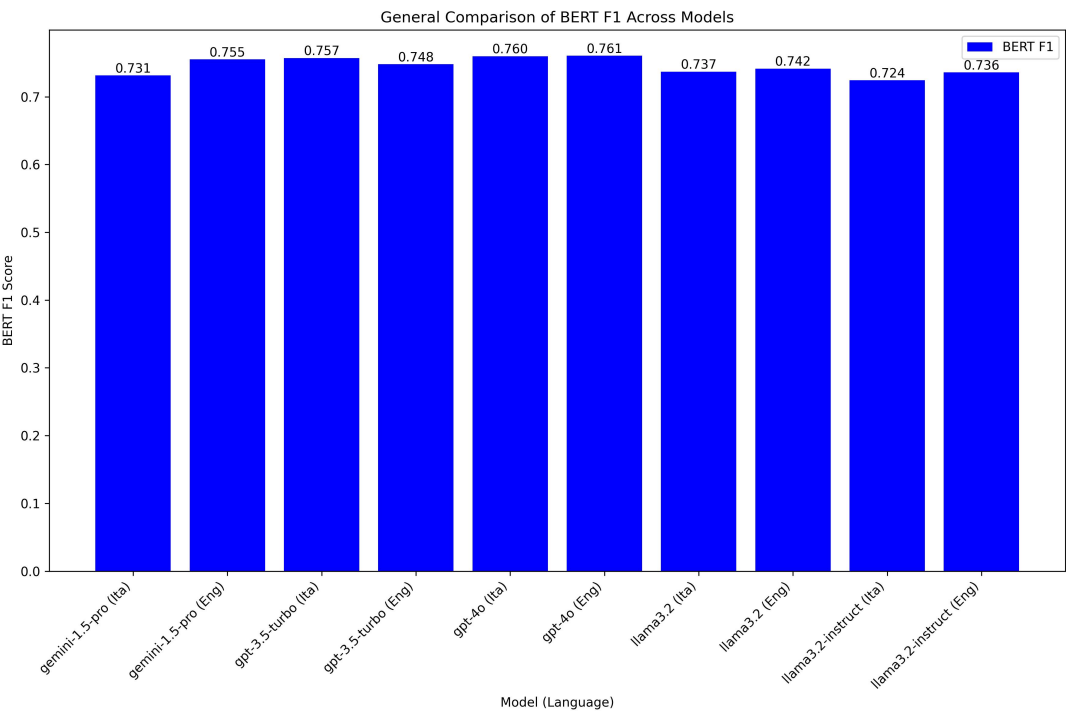


Figure B.8: General Comparison of BERT F1 Score.

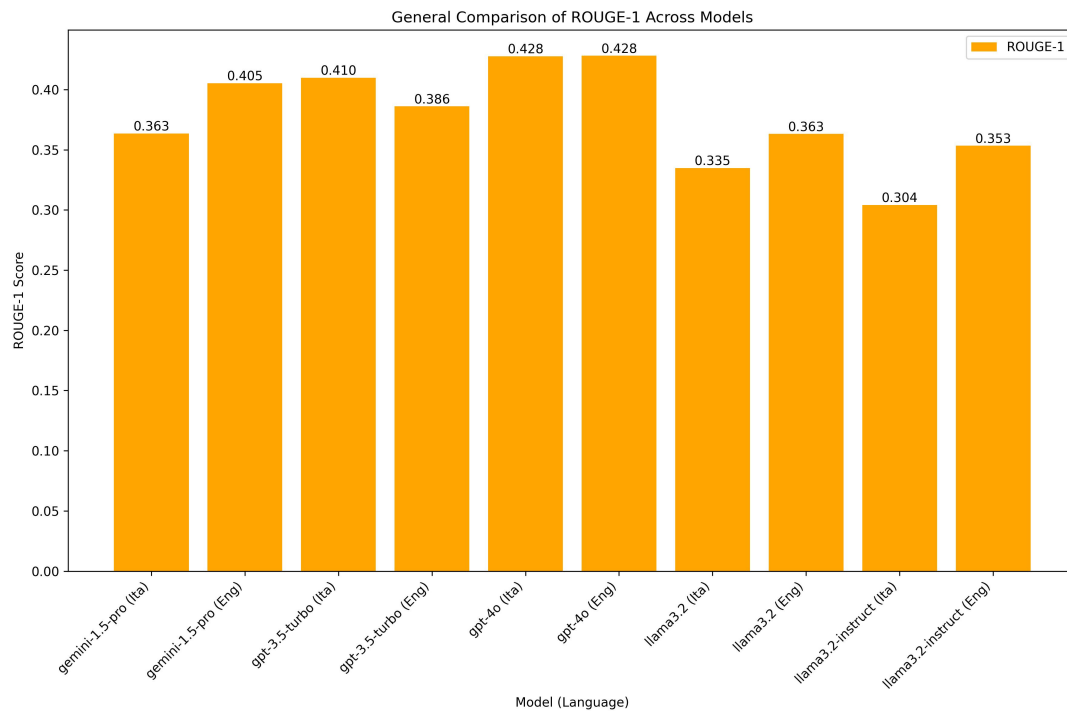


Figure B.9: General Comparison of ROUGE 1 Score.

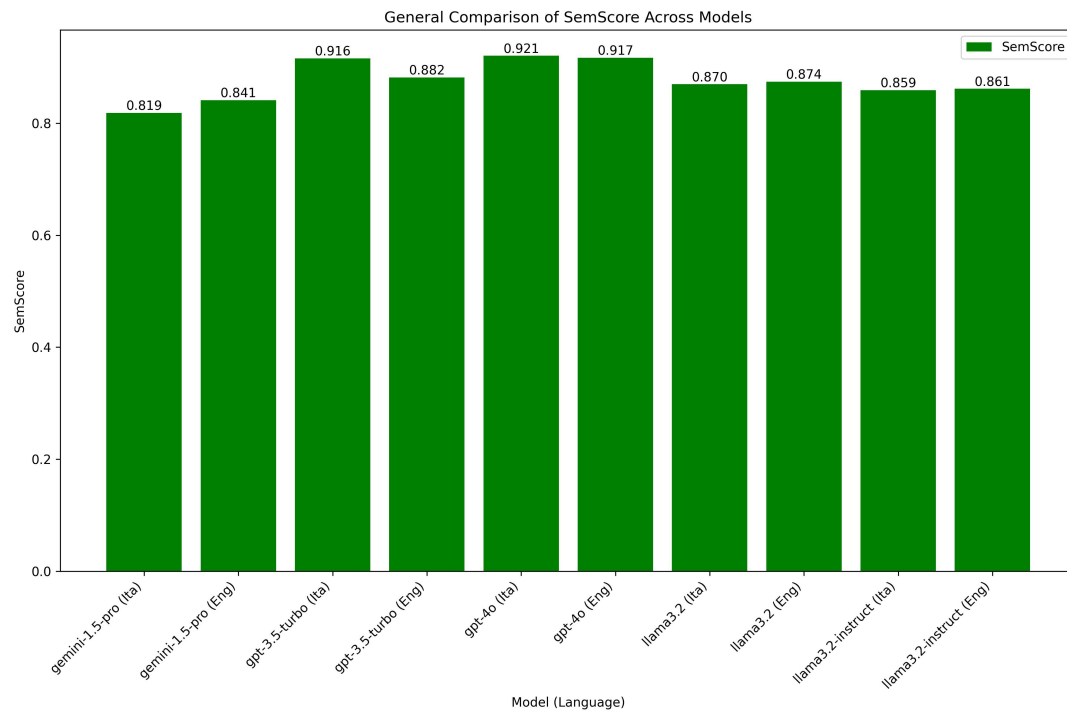


Figure B.10: General Comparison of SemScore.

## PoLL aggregation with Dawid-Skene against Majority Voting

The evaluation results presented in this section indicate that the Dawid-Skene (DS) method does not significantly improve agreement with the ground truth compared to Majority Voting (MV) on this dataset of the chatbot-arena conversations. While DS generally aims to refine aggregated labels by modeling annotator reliability, its limited improvement here suggests possible convergence issues. Thus, a reason for these results is that the DS method was not properly tuned for the dataset, leading to suboptimal results. Moreover, the nature of the task itself appears to be particularly challenging, making it difficult for individual models to perform reliably. The agreement scores below 50% and the negative correlation values indicate that the models exhibit performance that is worse than random chance.

| Method                    | Agreement with Ground Truth (%) |               |
|---------------------------|---------------------------------|---------------|
| Majority Voting           | 47.12                           |               |
| Dawid & Skene Multi-Class | 49.52                           |               |
| Metric                    | Majority Voting                 | Dawid & Skene |
| Cohen's Kappa             | -0.0577                         | -0.0096       |
| Pearson Correlation       | -0.0577                         | -0.0096       |
| Kendall-Tau               | -0.0577                         | -0.0096       |
| Exact Matches             | 196/416                         | 206/416       |

**Table B.1:** Results on chatbot-arena dataset for a PoLL of different foundation models: "GPT": "openai-community/gpt2", "Llama": "meta-llama/Llama-3.2-3B", "Gemma": "google/gemma-2-2b"

| Method                    | Agreement with Ground Truth (%) |               |
|---------------------------|---------------------------------|---------------|
| Majority Voting           | 47.71                           |               |
| Dawid & Skene Multi-Class | 43.12                           |               |
| Metric                    | Majority Voting                 | Dawid & Skene |
| Cohen's Kappa             | -0.0452                         | -0.1438       |
| Pearson Correlation       | -0.0452                         | -0.1496       |
| Kendall-Tau               | -0.0452                         | -0.1496       |
| Exact Matches             | 104/218                         | 94/218        |

**Table B.2:** PoLL of much more different foundation models: "GPT": "openai-community/gpt2", "Llama": "meta-llama/Llama-3.2-3B", "Gemma": "google/gemma-2-2b", "Mixtral": "mistralai/Mixtral-8x7B", "MiniGptBased": "model/mini-gpt", "DistilBERT": "huggingface/distilbert", "MiniSBERT": "model/mini-sbert", "DeepHaiku": "model/deep-haiku", "PalmBonsai": "model/palm-bonsai"

In conclusion, the results highlight the importance of better tuning DS parameters and evaluating also on other datasets. Additionally, improving the quality and diversity of the underlying models could help achieve more reliable outputs in future evaluations.



# List of Figures

2.1 **Naive RAG [15]:** *example of a RAG process from the 'RAG survey' paper, showing the main steps of this technique. The user gives as input a query, which the system is not able to answer to without external knowledge, thus, the RAG process becomes fundamental. The external knowledge, made by documents, is spitted in chunks encoded into vectors and the chunk vectors are indexed and stored in a vector database. When the user query arrives to the system, the RAG embeds the query with the same embedding of the chunks and extracts the most similar ones based on semantic similarity. Then, the initial question and the chunks are combined together obtaining a prompt that is fed to the LLM to generate the final answer. Thanks to the RAG process, the model now knows the answer and can give a response to the user.* . . . . . 14

3.1 **RAG Techniques [17]:** *basic and advanced RAG techniques that can be used to improve both the pre-retrieval and the post-retrieval process, enhancing the system as a whole, getting better performance and also improving the efficiency of the pipeline.* . . . . . 20

4.1 **Process for creating an augmented textual dataset:** *Raw documents are processed to extract both textual content and embedded images, which are converted into descriptive text using an image-to-text AI model and then integrated to form a comprehensive textual dataset, enhancing the original document's information* . . . . . 29

4.2 **Data augmentation process:** *Passing to Gemini the extracted image and the simplest prompt, the result is a good description of the main parts of the interface screen, adding to the textual data precise terms and labels from the software.* . . . . . 31

- 4.3 **Answer comparison with and without data augmentation:** *Following the same use case of the previous figure, there is a noticeable improvement of the obtained result, due to the introduction of the image description in the textual content of the documents, having an exact terminology match between the answer and the software interface, and obtaining a more precise response, having an improved alignment with user needs, guiding the user to the understanding of the functionality of the mentioned flag, and ensuring that the user can activate it, specifying the precise checkbox.* . . . 32
- 4.4 **Textual dataset main preprocessing steps:** *a simple, but partial preprocessing of the dataset, performing only the highlighted steps, removing the index, unwanted patterns, and normalizing the white spaces.* . . . . . 33
- 4.5 **Textual dataset simple preprocessing pipeline:** *differently from the partial simple preprocessing pipeline, all the preprocessing steps showed in the figure are now executed, adding in fact the removal of unstructured tables and flattening the lists, normalizing even more the textual content.* . 34
- 4.6 **Advanced textual dataset pre-processing pipeline:** *using a much more complex preprocessing pipeline, the previous result is further processed, trying to remove additional unwanted patterns, correct misspellings, eliminate punctuation, and convert all text to lowercase. In general, it is important to note that the punctuation removal may be optional depending on the chunking strategy. For instance, if sentence splitting is part of the chunking process, retaining punctuation could be essential. Therefore, this step is highlighted, because it should be carefully considered based on the subsequent stages of the application pipeline.* . . . . . 35
- 4.7 **Pipeline for vector database creation.** *The preprocessed documents are split into smaller chunks using a chunking strategy, and each chunk is then embedded resulting in vectors that capture the semantic meaning of the chunks, that are stored in the vectorstore, enabling efficient retrieval in the RAG pipeline.* . . . . . 40
- 4.8 **Vectorstore and similarity search [25]:** *search over unstructured data to retrieve relevant chunks for user query based on semantic similarity.* . . 49
- 4.9 **Basic RAG pipeline:** *given the user question, the chosen information retrieval technique is used to extract from the vectorstore the most relevant chunks, then combined with the initial query and instructions to guide the LLM's response.* . . . . . 51

|      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |     |
|------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| 4.10 | <b>Advanced RAG:</b> <i>building upon the previous pipeline, while maintaining the same flow of information and crucial steps, this new RAG implementation incorporates other advanced techniques, specifically, the query rewriting step and the RRF algorithm which improves the relevance of the extracted information. The advanced RAG aims to an optimized generation based on more effective context, an improved prompt, and the selected LLM, in order to have a high-quality final answer while balancing cost and performance.</i> | 58  |
| 4.11 | <b>RAG-Fusion [37]:</b> <i>combines RAG and RRF to obtain a better context curation and reduce the 'lost in the middle' problem, giving first to the LLM the chunks with the highest RRF scores.</i>                                                                                                                                                                                                                                                                                                                                          | 63  |
| 5.1  | <b>Synthetic evaluation-set creation:</b> <i>the complete LLM-powered pipeline, employed to generate the best QA for the evaluation of the RAG-based chatbot, in the absence of a ground truth, to use for the system quality assessment.</i>                                                                                                                                                                                                                                                                                                 | 68  |
| 5.2  | <b>Generate Question-Answers pairs using Mixtral-8x7B-Instruct-v0.1:</b> <i>employing three different sampling strategies, random chunks from the preprocessed documents are provided to the Mixtral model with a specific prompt to generate human-like, useful QA pairs of increasing complexity.</i>                                                                                                                                                                                                                                       | 69  |
| 5.3  | <b>Critique-Agents with Mixtral and QA pair filtering by score:</b> <i>using four critique-agents, the quality of the QAs is assessed; QAs related to image descriptions are filtered out, and only those that achieve a good score for all the agents, based on the criteria of groundedness, relevance, and standalone clarity are retained for the subsequent steps of the pipeline.</i>                                                                                                                                                   | 71  |
| 5.4  | <b>Multistep final refinement with Gemini model as judge:</b> <i>additional quality control, using a different LLM as a judge, is necessary to reduce errors and biases from a single model, to balance perspectives, and obtain a high quality evaluation set.</i>                                                                                                                                                                                                                                                                           | 72  |
| B.1  | <i>BERT F1 Score - English prompt.</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        | 106 |
| B.2  | <i>ROUGE-1 Score - English prompt.</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        | 106 |
| B.3  | <i>SemScore - English prompt.</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             | 107 |
| B.4  | <i>BERT F1 Score - Italian prompt.</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        | 107 |
| B.5  | <i>ROUGE-1 Score - Italian prompt.</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        | 108 |
| B.6  | <i>SemScore - Italian prompt.</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             | 108 |
| B.7  | <i>Comparison of all metrics across different models and languages.</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                                       | 109 |
| B.8  | <i>General Comparison of BERT F1 Score.</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   | 109 |
| B.9  | <i>General Comparison of ROUGE 1 Score.</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   | 110 |
| B.10 | <i>General Comparison of SemScore.</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        | 110 |



## List of Tables

|      |                                                                                                                                                                                                                                                                                                                                                          |    |
|------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| 4.1  | <i>Summary of Preprocessing Pipeline Experiments and Results . . . . .</i>                                                                                                                                                                                                                                                                               | 38 |
| 4.2  | <i>Hyper-parameter tuning of Chunk Size and Overlap for Recursive Chunking</i>                                                                                                                                                                                                                                                                           | 43 |
| 4.3  | <i>Recursive and Semantic Character Text Splitting experiments and results with simple preprocessing, tuning dense k from 2 to 10. It compares the best results of the two chunking techniques in the last table, confirming k = 4 as most effective value for retrieval. . . . .</i>                                                                    | 44 |
| 4.4  | <i>Comparison of Best Recursive and Semantic Chunking strategies on a mixed evaluation set, to reduce bias and obtain more reliable and robust results . .</i>                                                                                                                                                                                           | 46 |
| 4.5  | <i>Comparison between simple Naive RAG and Naive RAG with the previously explained improvements: advanced indexing using data augmentation and simple preprocessing, selecting the chunking strategy with associated hyper-parameter tuning; and improved retrieval by fine tuning the number of extracted chunks to pass to the LLM. . . . .</i>        | 52 |
| 4.6  | <i>Performance results of different retrievers, evaluated using the downstream metrics that best align with human judgment in the specific scenario. . . .</i>                                                                                                                                                                                           | 54 |
| 4.7  | <i>Performance comparison of different LLMs on Zero-shot prompting on small test set: GPT-4o model with Italian prompt is the combination that yields the best results. . . . .</i>                                                                                                                                                                      | 60 |
| 4.8  | <i>Comparison of Zero-shot, One-shot and Few-shot prompting using GPT-4o with Italian prompt: introducing few-shot prompting leads to an upgrade, but adding too many few-shot examples does not result in consistent improvements across the evaluated metrics, having excessive input tokens without significant performance enhancements. . . . .</i> | 61 |
| 4.9  | <i>Performance comparison of different RAG implementations, focusing on the impact of adding question rewriting to the pipeline pre-retrieval process.</i>                                                                                                                                                                                               | 62 |
| 4.10 | <i>Performance comparison of different advanced combinations of retrieval strategies, using question rewriting, a filtering threshold, and RRF. . . . .</i>                                                                                                                                                                                              | 64 |
| 4.11 | <i>Performance comparison of different LLM models for the final output generation, with diverse retrieval implementations for the RAG pipeline. . . .</i>                                                                                                                                                                                                | 65 |

- B.1 *Results on chatbot-arena dataset for a PoLL of different foundation models: "GPT": "openai-community/gpt2", "Llama": "meta-llama/Llama-3.2-3B", "Gemma": "google/gemma-2-2b" . . . . . 111*
- B.2 *PoLL of much more different foundation models: "GPT": "openai-community/gpt2", "Llama": "meta-llama/Llama-3.2-3B", "Gemma": "google/gemma-2-2b", "Mixtral": "mistralai/Mixtral-8x7B", "MiniGptBased": "model/mini-gpt", "DistilBERT": "huggingface/distilbert", "MiniSBERT": "model/mini-sbert", "DeepHaiku": "model/deep-haiku", "PalmBonsai": "model/palm-bonsai" . 111*