UNIVERSITÀ POLITECNICA DELLE MARCHE

FACOLTÀ DI ECONOMIA "GIORGIO FUÀ"

Bachelor's degree in Digital Economics and Business

DEVELOPMENT OF A RETRIEVAL AUGMENTED GENERATION SYSTEM FOR CUSTOMER REVIEWS

Supervisor:
Prof. Alex Mircoli

Thesis Candidate:
Alina Iuorio

Academic Year 2025/2026

INDEX

CHAPTER 1
INTRODUCTION

Thanks to the advent of Machine Learning and Artificial Intelligence of last decade, the corporate scenario has changed considerably. Due to the development of both, corporate tasks that were before complex and required huge amount of time, are nowadays more effective and efficient. The advent of technology is a crucial advantage for managing massive unstructured data for decision-making processes in business information systems.

Despite Artificial Intelligence offers optimal language and automation capabilities for processing data, the integration of Large Language Models remains limited by knowledge completeness and lack of context accuracy.

To bridge this gap, **Retrieval Augmented Generation** (RAG) emerges as a vital solution to transform unreliable, out of context and inaccurate models, into a system that aims to enhance the accuracy and relevance of the generated content, by integrating external data retrieval into text generation processes, providing an output that is consistent with the specific field analysed.

While the applications of a RAG system are multiple, especially for unstructured data and internal knowledge, this thesis specifically focuses on customer reviews for both B2B and B2C contexts, seeking to simplify data analysis and decision-making processes.

The case study is applying the RAG system to the **European Restauration** sector, since it represents a highly competitive industry and a vital macroeconomic driver. **Customer reviews** act as key performance indicators (KPI) when talking about a service, as the restauration, because they are parameters that influence decisions for both business actors and end consumers. Thanks to digitization, costumer reviews are now accessible and so, a valuable resource for this service, since:

- In **B2B**, customer reviews tell the company their actual positioning, so how they are actually perceived by their customers, providing crucial feedback and insight in order to improve their strategy. This is possible thanks to the RAG system that analyses all the aspects of a restaurant, for example food quality, location, customer service, average waiting time and other information useful to better understand and highlight strengths and pain points.

- In **B2C**, customer reviews act as virtual assistant for end users in order to help them find the restaurant that better fit their preferences. With the help of RAG system then, potential customers can actively search for a suitable restaurant avoiding waste of time but above all else, trustworthy information published by end users themselves, ensuring that their requests (as for example, vegetarian menu or live music) are satisfied.

These are the two pillars of how a RAG system can be applied to the restauration sector from both actors' perspective, emphasizing reliability and accuracy of the model. Companies should implement RAG system to optimize their customer experiences with AI, in order to develop a business strategy that is actually based on consistent and coherent data and increase customer satisfaction.

The rest of the thesis is organized as follows:

- Section 2: **Background**

  First of all, It will be discussed the theoretical background of the thesis concerning word embeddings and the Retrieval Augmented Generation, giving them definition, explaining processes and applications.

- Section 3: **RAG system for opinion mining**

  In this section, It will be discussed the practical framework of the Retrieval Augmented Generation, starting with the dataset explanation and all processes of data cleaning, and going on with the actual computation of the code on Python, matched by respective step-by-step interpretation.

- Section 4: **Conclusion**

  Eventually, It will be discussed the overall goal and analysis of the final outcome of the work, giving insight about the possible future work.

CHAPTER 2

FRAMEWORK

In the rapidly evolving landscape of AI applications, the ability to efficiently store, search, and retrieve information has become crucial for building intelligent systems, especially within enterprise environment and services involving customer experience. In the architectural landscape of modern Natural Language Processing (NLP), **text embeddings and Retrieval-Augmented Generation (RAG)** models do not operate in isolation; rather, they form a deeply codependent ecosystem.

This convergence solves a critical corporate challenge: while Large Language Models (LLMs) are linguistically advanced, their business integration is limited by data obsolescence and information hallucinations. To bypass the prohibitive capital costs of model retraining, Retrieval-Augmented Generation (RAG) decouples an AI's reasoning from its static parameters by pulling data dynamically from external sources.

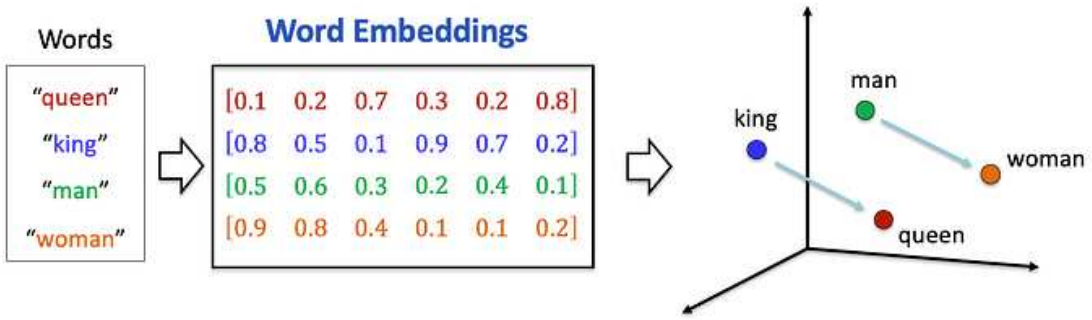
Crucially, this RAG workflow relies entirely on the mathematical precision of text embeddings [1]. Because raw human language cannot be processed directly for semantic search, embeddings translate unstructured data, for example like supplier catalogs or customer queries, into dense, multi-dimensional vector spaces where concepts are mapped by geometric

proximity. The RAG architecture then uses these vector coordinates to execute high-speed similarity matching, fetching the exact context needed to generate verified, accurate responses [1]. In digital business models, this synergy between embedding math and RAG retrieval transforms generative AI from an isolated, speculative tool into a high-precision enterprise asset, as for customer reviews management in restaurants service.

2.1 Word Embeddings

A key method in natural language processing (NLP) is text embedding, which seeks to compactly and meaningfully represent textual data. It is essential to many NLP tasks, including question answering, sentiment analysis, and information retrieval [1].

Word embeddings are a way of representing words as vectors in a multi-dimensional space, where the distance and direction between vectors reflect the similarity and relationships among the corresponding words [2]. By assigning numerical values to words based on their semantic similarities, word embedding helps neural network models understand context more efficiently. This approach reduces computational complexity and enhances model performance by preserving semantic information [3].



This is a graphical representation of the word embeddings processes.

Traditional enterprise databases generally rely on rigid setups like SQL tables or basic keyword search functions to organize records. Text embeddings handle things differently. They change how an information system processes unstructured data by converting raw text directly into a dynamic semantic map, which is technically known as a dense vector space.

From a management information perspective, you can basically think of this structure as a digital asset map where every document, invoice, or customer transcript receives its own multi-dimensional address. The system does not just match exact words anymore. Instead, it groups data entirely by operational context. Ideas that belong together naturally end up next to each other in the system. For instance, if an e-commerce guest searches a restaurant platform for "allergen-free choices," the system maps that query directly to the same neighbourhood as a kitchen manager's "gluten-free

ingredient sheet." It bypasses the old limits of exact keyword matching entirely.

For what concern the word embeddings processes itself, to deploy this at an enterprise scale, unstructured text has to move through a step-by-step pipeline. In practice, this functions exactly like a standard Extraction, Transformation, and Loading (ETL) workflow, just tailored for language:

- **Extraction and Tokenization:** The system ingests raw corporate documents, like massive wholesale supplier catalogs or customer service reviews, and cuts the text into smaller semantic pieces called tokens.
- **Transformation into Vectors:** These tokens feed directly into an AI encoder model. The encoder reads how words relate to the sentences around them, translating the qualitative text into a distinct numerical profile.
- **Indexing and Storage:** The final numbers are stored securely inside a specialized vector database component.

When a user submits a query, the system runs this exact pipeline in reverse. It calculates semantic similarity, matching the user's math coordinates against the database index to pull out matching background text in real time.

The applications of word embeddings are multiple in the context of business information systems. Turning qualitative language into a measurable, structured system asset opens up huge advantages across a company's value chain. In digital economics, this directly fixes the costly issues of internal search friction and information asymmetry. Modern business applications include:

- **Intelligent Knowledge Management:** This means replacing legacy, broken Ctrl+F portals with smart internal search systems that understand user intent. Employees can instantly navigate complex internal wikis, old sales receipts, and deep training manuals without needing exact keyword matches.
- **Customer Experience (CX) Automation:** Systems can automatically read, sort, and route incoming customer tickets, track customer mood trends from reviews, and supply precise background data to automated customer interfaces.
- **The Foundation for RAG Systems:** Embeddings act as the vital indexing layer that makes Retrieval-Augmented Generation work in the first place. If embeddings don't sort and clean the corporate knowledge layer first, a RAG pipeline has no way to locate the exact

background information required to back up automated business decisions.

Last two are crucial pillars of the development of the RAG system for customer reviews in European Restauration sector. [4][5]

2.2 Retrieval Augmented Generation (RAG)

While Large Language Models (LLMs) are revolutionary for processing text, deploying them directly into business workflows highlights severe parametric limitations. Standard models operate on static, historic datasets. Consequently, they lack access to real-time or updated information, making them generating confident but completely false statements known as hallucinations.

Here comes, as a solution, the **Retrieval Augmented Generation (RAG)** which stands out as one of the most practical techniques in contemporary enterprise AI. At its core, the architecture combines the data precision of retrieving real-time information from massive datasets with the computational reasoning power of large language models, resulting in system responses that are not just highly accurate, but deeply relevant to the end-user. Because of this structural reliability, RAG is currently transforming modern information systems, powering everything from

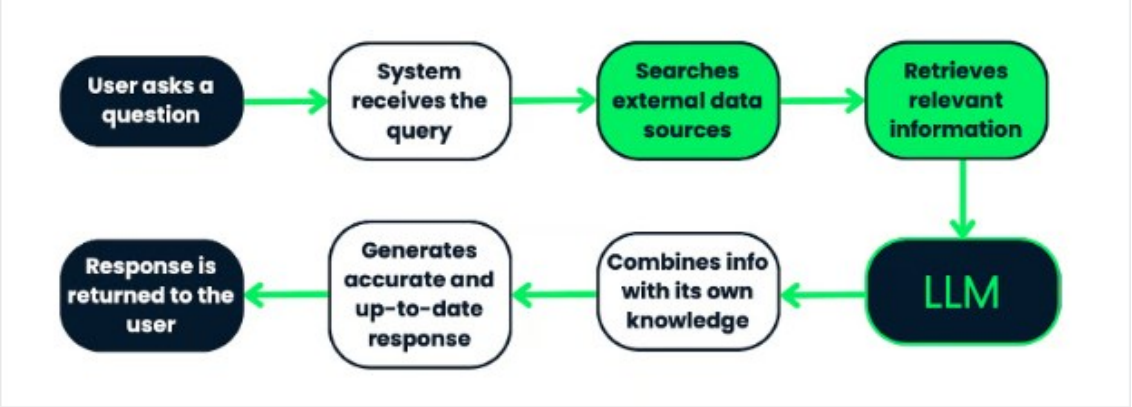
advanced corporate chatbots and semantic search engines to personalized content delivery platforms, for what concern the case study of this thesis.

As examined throughout this background chapter, RAG is an advanced method in natural language processing that fundamentally levels up what language models can achieve. Instead of just relying on what a model already knows from its historical training, RAG takes things further by pulling in new, relevant information from outside sources before generating a final response.

Here is how it works in practice, taking in account our case example: when a dining customer asks a specific question or searches for feedback, the system does not just guess based on the model's pre-learned data. It first goes out, searches through a massive dataset of restaurant customer reviews and service logs, grabs the most relevant bits of text, and feeds them directly into the language model. Equipped with both its built-in linguistic knowledge and this newly retrieved context, the model can create a response that is far more accurate and up to date.

By blending real-time retrieval with language generation, the final system responses are not just smart, but they are also strictly grounded in real, factual guest feedback. This architecture makes it the perfect solution for restaurant service applications, from automated customer support tools to

review-backed recommendation engines where getting the facts right and understanding user context really matters.



Workflow of a RAG system. It begins with a user query, which is processed by the system to search external data sources for relevant information. The retrieved information is then fed into an LLM, which combines it with its pre-existing knowledge to generate an accurate and up-to-date response. Finally, the response is returned to the user. This process ensures that responses are grounded in factual, contextually relevant data.

When building a RAG system, there are few essentials key components to take in account through the whole process.

The first step is getting our data ready, this function is carried on by:

- **Document Loaders**: These tools pull in data from various sources (text files, PDFs, databases…) They convert that info into a format the system can actually use. Basically, they make sure all the important data is ready and in the right shape for the next steps.

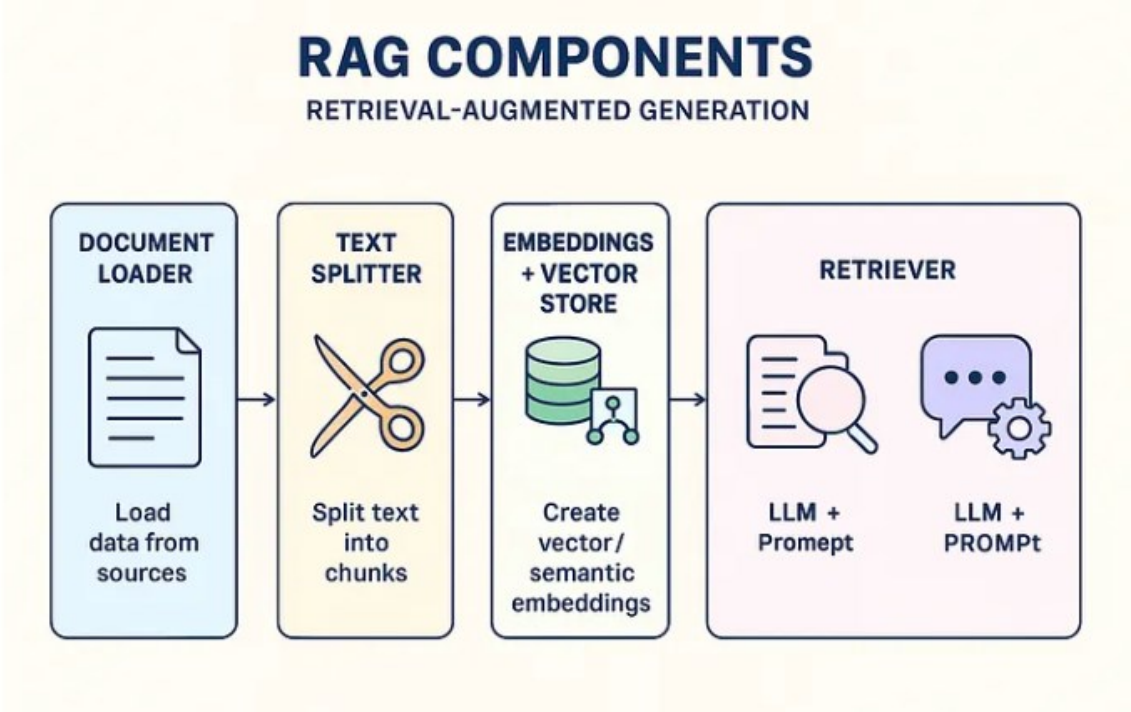
- **Text Splitting**: Once the data is loaded, it gets chopped into smaller chunks. This is super important because smaller pieces are easier to search through, and language models work better with bite-sized bits of info due to their processing limits.
- **Indexing**: After splitting, you need to organize the data. Indexing turns those text chunks into vector representations. This setup makes it easy and fast for the system to search through all that data and find what's most relevant to a user's query. This is the exact moment raw text transforms into an **embedding**.

Retrieval models are a crucial point of the RAG system. They are responsible for digging through all that indexed data to find what you need.

- **Vector Stores**: These are databases designed to handle those vector representations of the text chunks. They make searching truly efficient by using a method called vector similarity search, which compares the query to the stored vectors and pulls the best matches.
- **Retrievers**: These components do the actual searching. They take the user's query, convert it into a vector, and then search the vector store to find the most relevant data. Once they grab that info, it's passed along to the next step: generation.

The most relevant step to highlight is the one concerning generative models. Once the relevant data is retrieved, the generative models take over and produce a final response.

- **Language Models**: These models create the actual response, making sure it is coherent and fits the context. In a RAG system, they take both the retrieved data and their own internal knowledge to generate a response that's up-to-date and accurate.
- **Contextual Response Generation**: The generative model blends the user's question with the retrieved data to create a response that not only answers the question but also reflects the specific details from the relevant info it pulled.[6]



CHAPTER 3

RAG SYSTEM FOR OPINION MINING

Throughout this chapter, we will explore the actual process of the analysis done, starting with the description of data used, to the computation of the Python code for implementation and development of the RAG system applied to customer reviews in European restaurants sector.

Before giving a technical overview, it is interesting to highlight a complex natural language challenge that the implementation of the RAG system aims to address: the **opinion mining**, which is an important task in text mining and natural language processing that extracts structured sentiment information from reviews. Consumer feedback is a highly valuable asset for digital business strategy, but since restaurant reviews are typically short, nuanced, and filled with implicit sentiment that are completely missed by traditional keyword-based searches, it represents a challenge in retrieving reliable information. This is where RAG system plays a crucial role.[7]

3.1 Dataset

The starting point of the practical framework of this thesis is about the data chosen for the implementation of the RAG system, which is a crucial point since It is a baseline of the whole project.

As introduced before, data that are chosen, refer to the restauration sector, which is a business area exposed to the customer reviews, and introducing a RAG system as a problem-solver tool for both business actors is something truly valuable nowadays.

The initial idea was to gather customer reviews from TripAdvisor website through the process of web scraping, but since this idea was not pursued, it was ultimately decided to use a Kaggle dataset obtained by scaping Trip Advisor, called **TA_restaurants_curated.csv.** [8]

The dataset contain restaurants information for 31 cities in Europe: Amsterdam (NL), Athens (GR) , Barcelona (ES) , Berlin (DE), Bratislava (SK), Bruxelles (BE), Budapest (HU), Copenhagen (DK), Dublin (IE), Edinburgh (UK), Geneva (CH), Helsinki (FI), Hamburg (DE), Krakow (PL), Lisbon (PT), Ljubljana (SI), London (UK), Luxembourg (LU), Madrid (ES), Lyon (FR), Milan (IT), Munich (DE), Oporto (PT), Oslo (NO), Paris (FR), Prague (CZ), Rome (IT), Stockholm (SE), Vienna (AT), Warsaw (PL), Zurich (CH).

The data is a .csv file that contains 125433 entries (restaurants). It is structured as follow:

2, La Rive, Amsterdam, "['Mediterranean', 'French', 'International', 'European','Vegetarian Friendly', 'Vegan Options']",3.0,4.5,$$$$,567.0,"[['Satisfaction', 'Delicious old school restaurant'], ['01/04/2018', '01/04/2018']]",/Restaurant_Review-g188590-d696959-Reviews-La_Rive-Amsterdam_North_Holland_Province.html,d696959

- **Name**: name of the restaurant
- **City**: city location of the restaurant
- **Cuisine Style**: cuisine style(s) of the restaurant, in a Python list object (94 046 non-null)
- **Ranking**: rank of the restaurant among the total number of restaurants in the city
- **Rating**: rate of the restaurant on a scale from 1 to 5
- **Price Range**: price range of the restaurant among 3 categories
- **Number of Reviews**: number of reviews that customers have let to the restaurant

- **Reviews**: 2 reviews that are displayed on the restaurants scrolling page of the city, and the second dates when these reviews were written
- **URL_TA**: part of the URL of the detailed restaurant page that comes after 'www.tripadvisor.com'
- **ID_TA**: identification of the restaurant in the TA database constructed a one letter and a number

Since some variables are not useful for the analysis, we decided to select only those relevant for the final objective, so Name, City, Cuisine style, and Reviews. Taking in account that the goal of this initial stage was to create a pdf which contained the relevant variables of the dataset and to load it into the RAG code, in order to perform the RAG process and retrieving restaurant information from this pdf.

The data cleaning of the csv has been performed through a Python code in Pycharm. Thanks to the use of pandas, a library for data manipulation, we removed all unnecessary variables, keeping just Name, City, Cuisine style and Reviews.

```
import pandas as pd
```

We focused especially on this last variable, because It contained two reviews each with their respective release dates, we decided to eliminate dates and to isolate each review in a single line. In the end, as an output we obtained a pdf file of cleaned data ready to be loaded into our RAG code.

These are main steps of data cleaning:

```
# 1. Load the dataset (make sure the file path matches your PyCharm project setup
file_path = "TA_restaurants_curated.csv"
df = pd.read_csv(file_path)
# 2. Select the relevant columns from the original dataset
target_columns = ["Name", "City", "Cuisine Style", "Reviews"]
df_filtered = df[target_columns].copy()
# 3. Clean and convert Cuisine Style into a standard clean string (e.g., "French,
df_filtered["Cuisine Style"] = (
    df_filtered["Cuisine Style"]
    .apply(extract_and_clean_list_column)
    .apply(lambda x: ", ".join(x) if isinstance(x, list) else "")
)
# 4. Clean and split the reviews into lists of individual phrases
df_filtered["Reviews"] = df_filtered["Reviews"].apply(
    extract_and_clean_list_column
)
# 5. Explode the rows to isolate each single review on its own row
df_final = df_filtered.explode("Reviews")
```

The output obtained:

Martine of Martine's Table	Amsterdam	French, Dutch, European	Just like home
Martine of Martine's Table	Amsterdam	French, Dutch, European	A Warm Welcome to Wintry Amsterdam
De Silveren Spiegel	Amsterdam	Dutch, European, Vegetarian Friendly, Gluten Free Options	Great food and staff
De Silveren Spiegel	Amsterdam	Dutch, European, Vegetarian Friendly, Gluten Free Options	just perfect

3.2 RAG System Implementation

After the step of data cleaning and data preparation, we stepped into the crucial part of the process: the implementation of the RAG system in

Python. We performed the computation as well as the data cleaning through Pycharm.

Building upon the theoretical principles of data retrieval and language models introduced in the previous chapter, this section details the complete technical implementation of the localized Retrieval Augmented Generation (RAG) system. The explicit focus of this data pipeline is opinion mining within the European restaurant sector. By automating the extraction and analysis of customer reviews, this pipeline transforms raw, unstructured data into actionable digital assets capable of driving business decision-making and market research.

The initial step includes the set up and global environment variables: the pipeline begins by configuring the execution environment, importing necessary frameworks, and defining global system parameters.

The script heavily leverages **LangChain**, a library designed to link complex data workflows together. It also utilizes **Ollama**, an environment that allows open-source artificial intelligence models to be run completely locally on consumer hardware. Going more in detail:

- **DirectoryLoader & UnstructuredPDFLoader**: Automatically scan files and extract unstructured text from your source folder (univpm_pdfs).

- **RecursiveCharacterTextSplitter**: Divides the continuous extracted text into precise, manageable blocks (chunk_size=800). This function is for text segmentation
- **OllamaEmbeddings**: Converts raw text segments into multi-dimensional numerical coordinate arrays (vectors) using the qwen3-embedding:8b model. This allows for semantic search based on conceptual meaning rather than matching exact keyword strings.
- **Chroma**: A specialized, local vector database that permanently records these coordinates on disk. When the query is executed, it instantly calculates mathematical similarity, isolating and fetching only the top 4 (k=4 as an example) most relevant data.
- **ChatOllama**: Runs your core 14-billion parameter language model (deepseek-r1:14b) completely locally on your hardware. It acts as the pipeline's reasoning engine with zero variable API fees, keeping your consumer review dataset completely private.
- **ChatPromptTemplate**: Restricts the AI's creative behavior by framing the query with strict system rules, completely removing the corporate risk of AI hallucinations.
- **RunnablePassthrough & StrOutputParser**: Automates the data workflow. These modules construct a clean data assembly line that

processes parallel tasks simultaneously and streams the final, clean answer onto the user's dashboard token-by-token in real-time. [9]

For what concern the RAG implementation itself, after installing libraries the process follows these main steps:

1. **Step: Setting up our offline workspace and models**

```
DB_PATH = "./chroma_db_data"
CSV_RESTAURANTS_PATH = (
    "restaurants_cleandata.pdf") # path of PDF documents
LLM_MODEL = "deepseek-r1:14b"
EMBEDDING_MODEL = "qwen3-embedding:8b"
DOCS_FOLDER="univpm_pdfs"
```

This initial section establishes the system paths and global variables. It declares the target file location (univpm_pdfs/restaurants_cleandata.pdf), sets up the directory for the local database (./chroma_db_data), and specifies the open-source embedding and reasoning models (qwen3-embedding:8b and deepseek-r1:14b) that will be executed locally via the Ollama framework.

This configuration forms the security and financial baseline of the architecture. By running the language model and embedding engine completely offline, it eliminates the unpredictable subscription and API usage costs associated with commercial clouds. Most importantly for market research, it guarantees absolute data privacy (GDPR): sensitive,

proprietary customer review datasets remain fully contained within the local corporate hardware.

2. **Step: Loading PDFs cleanly without scrambling data**

```
print("\nPDF INGESTION THROUGH OCR")
loader_kwargs = {
    "strategy": "hi_res",
    "languages": ["ita", "eng"]
}
loader = DirectoryLoader(
    DOCS_FOLDER,
    glob="*.pdf",
    loader_cls=UnstructuredPDFLoader,
    loader_kwargs=loader_kwargs,  # OCR options
    show_progress=True
)
docs = loader.load()
```

The program launches a DirectoryLoader to sweep the target folder for PDFs, wrapping around UnstructuredPDFLoader. It injects a layout-parsing strategy (hi_res) and dual-language dictionaries (["ita", "eng"]) to process the document visually and linguistically before loading it into system memory.

In standard text processing, converting tables or spreadsheets into PDFs breaks the data structure. Ordinary PDF readers extract text horizontally, causing side-by-side columns to blend together into an unreadable mess. The high-resolution strategy prevents this error by analyzing the layout of rows and borders, ensuring each text review stays perfectly linked to its

correct restaurant name and location. Concurrently, multi-language mapping prevents text distortion when processing regional restaurant names, European locales, or localized customer phrasing (for example, mixing Italian and English terms).

3. Step: Cutting long reviews into readable pieces

```
# splitting
print("Document splitting")
text_splitter = RecursiveCharacterTextSplitter(chunk_size=800, chunk_overlap=150)
splits = text_splitter.split_documents(docs)
print(f"Created {len(splits)} text chunks.")
```

This step uses a RecursiveCharacterTextSplitter to slice the extracted text into smaller, bounded strings. It establishes a strict maximum ceiling for each segment (chunk_size=800) and a rolling boundary overlap (chunk_overlap=150).

Large Language Models operate under strict context limitations and process data less effectively when fed massive blocks of raw text at once. Breaking the document into 800-character units ensures the text fragments remain digestible for the AI. The 150-character overlap acts as a safety margin by copying trailing text over to the next segment. This prevents critical review comments from being awkwardly cut mid-sentence at a document boundary, preserving context continuity.

4. Step: Translating words into smart concepts

```
# embedding creation and saving
print("Embedding creation and saving on ChromaDB")
Chroma.from_documents(
    documents=splits,
    embedding=OllamaEmbeddings(model=EMBEDDING_MODEL),
    persist_directory=DB_PATH
)
print("Database created!")
return True
```

The code passes the text chunks to OllamaEmbeddings running the Qwen3 model, translating the words into mathematical coordinates (dense vectors). These coordinates, along with their matching raw text fragments, are written permanently onto the hard drive using the Chroma database library. Traditional databases search for records using rigid keyword lookups. If an analyst queries the database for "good location," a standard search will miss reviews containing phrases like "stunning view" or "central spot." Vector embeddings solve this limitation by translating the underlying meaning of the words into numerical coordinates. This allows the database to run conceptual searches, instantly surfacing hidden trends and relevant customer insights regardless of the exact vocabulary the reviewer used.

5. **Step: Isolating only the most relevant feedback**

```
def query_rag(question):  1 usage
    print(f"\n--- Question: {question} ---")
    vector_db = Chroma(
        persist_directory=DB_PATH,
        embedding_function=OllamaEmbeddings(model=EMBEDDING_MODEL)
    )
    retriever = vector_db.as_retriever(search_kwargs={"k": 4})
    llm = ChatOllama(model=LLM_MODEL)
```

When a user inputs a natural language question, the function opens the saved Chroma database. It initializes a retriever configured with search_kwargs={"k": 4} to isolate the top 4 closest vector matches and readies the local DeepSeek model engine.

This step functions as an automated data noise filter. Instead of forcing the AI model to read through the entire European restaurant dataset, the retriever matches the query coordinate and instantly pulls only the 4 most conceptually relevant text snippets. For example, if an analyst asks a question specific to Amsterdam, the database isolates only the text blocks involving Amsterdam, filtering out unrelated records from other markets and focusing the AI's attention purely on the data needed to answer the question.

6. **Step: Getting real, hallucination-free answers**

```
template = """Generate an answer based on the context only.
Context: {context}
Question: {question}
Answer:"""
prompt = ChatPromptTemplate.from_template(template)

def format_docs(docs):
    return "\n\n".join([d.page_content for d in docs])

rag_chain = (
        {"context": retriever | format_docs, "question": RunnablePassthrough()}
        | prompt
        | llm
        | StrOutputParser()
)

for chunk in rag_chain.stream(question):
    print(chunk, end="", flush=True)
print("\n")
```

Using LangChain Expression Language (LCEL), the code builds a sequential processing line (rag_chain) using the pipe (|) operator. It packages the question and the retrieved text snippets ({context}) into a strict template structure, routes the template payload to the DeepSeek model, and parses the text dynamically using .stream().

Unconstrained language models are prone to "hallucinations", so inventing plausible-sounding facts or blending general internet training data with your actual dataset metrics. The prompt template enforces a strict deterministic rule: "Generate an answer based on the context only." This guardrail shuts down the model's creative liberties, forcing it to act purely as an objective summarizer of your verified source data sheet. Finally, calling .stream() flushes individual words onto the display in real-time as

they are processed by DeepSeek, eliminating system lag and allowing analysts to view actionable business insights immediately.

CHAPTER 4

CONCLUSION

The goal of the project was to implement and develop a Retrieval Augmented Generation system for customer reviews of Europeans restaurants, useful for both B2B and B2C actors. Thus, this thesis aimed to create a problem-solver tool that could be used by restaurant businesses for managing their strategy based on actual data, and by end-users for finding restaurants that fit their preferences. Due to the advent of AI and Machine Learning, the RAG system had the role of a corporate asset for improving customer experience, acting as a gained advantage on both sides.

During the implementation of the RAG itself, we first did an overview of the theoretical framework, stressing the importance of the word embeddings and all the components of the RAG process. We continued analysing the dataset and main steps of the Python code, explain both the technical aspects and the advantages of what done. In conclusion, we highlighted the opinion mining challenges and how RAG acted as a solution to face them.

4.1 Limits and Critical Points

During the development of this project, some pain points emerged, regarding data and the chosen LLM.

The initial idea was to collect data directly from Trip Advisor through web scraping, but since we failed it, in the end data was taken from a Kaggle dataset, that even if adapted, it was limited in the number of reviews and risked being outdated. For what concerns the LLM chosen, the deepseek could be not powerful enough.

4.2 Future work

This thesis is a good starting point for improving the project. Some inputs that could be interesting to develop in the future, work on the critical points of it. Some inputs for future work:

- Data accuracy could be refined by collecting data directly from web scraping Trip Advisor.
- DeepSeek LLM, as ollama, should be replaced by more powerful one, in order to better manage greater amount from unstructured data
- Provide consumers an user-interface that enables people to use the RAG system in real life, to gain actual advantage for B2C sector.

BIBLIOGRAPHY

[1] I. O. William and M. Altamimi, “Text Embedding Implementation Using Retrieval Augmented Generation (RAG) Model Combined With Large Language Model,” in Proc. 3rd Int. Conf. Eng., Natural and Social Sciences (ICENSOS), Konya, Turkey, May 16–17, 2024, pp. 667–675.

[2] https://www.ibm.com/think/topics/word-embeddings

[3] https://www.hpe.com/emea_europe/en/what-is/word-embedding.html

[4] M. Arslan, S. Munawar, and C. Cruz, “Business insights using RAG–LLMs: a review and case study,” Journal of Decision Systems, pp. 1–30, 2024, doi: 10.1080/12460125.2024.2410040.

[5] E. Karakurt, “Retrieval-Augmented Generation (RAG) and Large Language Models (LLMs) for Enterprise Knowledge Management and Document Automation: A Systematic Literature Review,” Applied Sciences, vol. 16, no. 1, p. 368, 2026.

[6] Building a RAG System with LangChain and FastAPI: From Development to Production | DataCamp

[7] Zhou, S., Qiu, J., Lin, L., Siyu, W., & Xu, Y. (2026). ROQuA: A RAG-based opinion question-answering framework for e-commerce reviews. *Electronic Commerce Research and Applications*, *76*, 101576. https://doi.org/10.1016/j.elerap.2026.101576

[8] TripAdvisor Restaurants Info for 31 Euro-Cities https://www.kaggle.com/datasets/damienbeneschi/krakow-ta-restaurans-data-raw

[9] Chase, H. (2022). LangChain: Building applications with LLMs through composability

https://github.com/langchain-ai/langchain