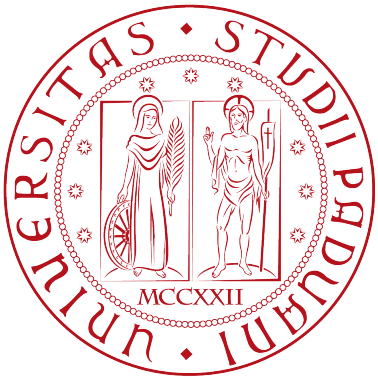
University of Padova

Department of Mathematics "Tullio Levi-Civita"

Master's in Computer Science



Enhancing Corporate Document Management Systems with AI: Leveraging Embeddings and Semantic Search for Efficient Retrieval and Chat-based Assistance

Master Thesis

Supervisor
Prof. Fabio Aiolli

Co-supervisor
Dr. Luca Bergamin

Student
Nazanin Ghorbani

A.Y. 2023-2024

To my family

Abstract

Recent advancements in large language models (LLMs) and Retrieval-Augmented Generation (RAG) techniques have revolutionized information management across industries, enabling the development of intelligent, efficient, and contextually-aware document interaction systems. This thesis explores the application of these technologies within the domain of corporate document management, presenting a novel chatbot solution designed to enhance document retrieval and provide accurate, context-driven assistance to users.

Building on this robust retrieval framework, the Retrieval-Augmented Generation (RAG) approach is used to integrate the best embedding model with five distinct LLMs. These models generate context-aware responses, with their performance evaluated based on response quality and alignment with user expectations. Metrics such as generation time are also analyzed to assess system efficiency.

This thesis demonstrates how the integration of LLMs, RAG, and advanced embedding techniques can transform corporate document management by providing reliable, scalable, and responsive access to knowledge. By detailing the system's architecture, methodology, evaluative metrics, and performance benchmarks, this work highlights the potential for deploying LLM-powered, RAG-driven solutions that enable efficient and contextually relevant user interactions across industries.

Acknowledgements

This thesis wouldn't have been possible without the people who have guided, supported, and inspired me along the way. I am deeply grateful to each one of you.

First I want to thank my professor, Prof. Fabio Aiolli, for his invaluable guidance and expertise throughout this project. My sincerest thanks to Roberto Onnis at FlossLab for his mentorship and constructive feedback, which have greatly contributed to the quality of this work. My thanks also extend to Dr. Luca Bergamin for his thoughtful insights and academic support during the development of this thesis.

I acknowledge the Department of Mathematics at the University of Padova for providing a stimulating academic environment and the resources necessary for pursuing this research and a special thanks also to FlossLab for providing me with the opportunity to work on this exciting project and for fostering an enriching environment for learning and development..

To my family—Mom, Dad, my siblings and my brother-in-law Mattias—thank you for being my rock. Your love, sacrifices, and constant belief in me have given me the strength to reach this milestone. I truly couldn't have done it without you.

I am also deeply thankful for the friends who have stood by me through this journey, offering encouragement, advice, and moments of joy. Finally, I want to thank my boyfriend, Andrea, for his patience and constant support, which have been a source of strength throughout this process.

Padova, December, 2024 Nazanin Ghorbani

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter outlines the motivation and objectives of the research, provides contextual information, and presents the research goals, followed by an overview of the thesis structure.

Flosslab, founded in 2007 as the University of Cagliari's first spin-off, specializes in providing cutting-edge software solutions tailored for both public and private sectors. The company's expertise spans key domains such as process digitalization, document management, web development, and blockchain technologies. In 2018, Flosslab became part of the Net Service SpA Group, significantly enhancing its capacity to deliver scalable and innovative digital solutions.

A cornerstone of Flosslab's operations is its commitment to research and development. Backed by Sardegna Ricerche, the company has established a robust innovation program aimed at integrating advanced technologies into its solutions. Notable initiatives include **CAFCHA**, which employs blockchain to ensure traceability and safety in the agro-food industry, and **CASCO**, a project focused on enhancing smart contract security by identifying vulnerabilities and tracking token-related data within blockchain networks.

In keeping with its forward-looking approach, Flosslab has recently expanded its focus to explore the transformative potential of Artificial Intelligence (AI). This initiative includes employing Natural Language Processing (NLP) tools to enhance document management systems by enabling semantic understanding, context-aware search, and more intuitive data interaction. By leveraging these capabilities, Flosslab aims to improve operational efficiency and make information more accessible and actionable for users.

My collaboration with FlossLab aligns with this vision, as I contributed to the development of an advanced tool designed to optimize document retrieval and management. This project leverages **Retrieval-Augmented Generation (RAG)**, a cutting-edge AI methodology introduced by [1], to enhance the efficiency and effectiveness of accessing corporate documents. By incorporating AI-driven techniques such as semantic search and chat-based assistance, the solution tackles the challenges of managing extensive information repositories, providing streamlined and intuitive access to critical data.

This thesis documents the research, development process, and outcomes of this collaboration. It evaluates the potential of RAG and AI technologies in enhancing corporate document management systems, ultimately contributing to Flosslab's mission of delivering innovative, technology-driven solutions.

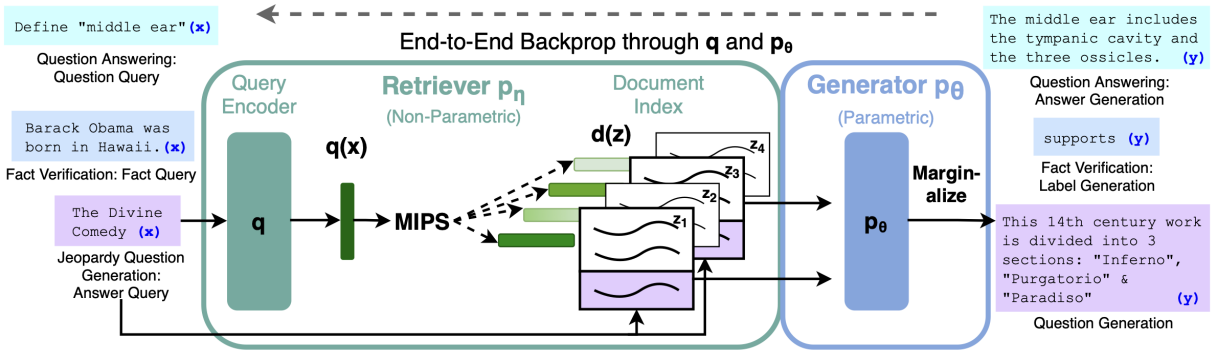


Figure 1.1: Illustration of the Retrieval-Augmented Generation (RAG) framework, adapted from Lewis et al. (2020) [1]. The framework combines a pre-trained retriever (Query Encoder + Document Index) with a generative model (Generator). For a given query x, the retriever uses Maximum Inner Product Search (MIPS) to identify the top-K relevant documents (z_i) from the document index. The generator then produces the final output (y) by marginalizing over the retrieved documents, optimizing retrieval and generation tasks in an end-to-end manner. This approach enables the efficient resolution of knowledge-intensive tasks, such as question answering and semantic search.

1.1 Scope

In today's data-driven business environment, organizations handle extensive volumes of documents, such as contracts, reports, policies, and other critical information. Traditional document management systems often face limitations in efficiently retrieving relevant information, particularly as the volume and complexity of documents increase [2]. Keyword-based search systems often fail to capture contextual nuances and semantic relationships, leading to reduced productivity and potential errors when employees need to locate critical information. For example, a doctor searching for "diabetes treatment" in a keyword-based system might receive irrelevant results, such as outdated protocols or unrelated news articles, instead of the latest context-specific clinical guidelines. This inefficiency underscores the importance of semantic search systems, which leverage contextual understanding to deliver accurate and relevant information.

This research addresses these inefficiencies by leveraging Artificial Intelligence (AI) to advance document management capabilities. Specifically, it explores how embedding models, semantic search, and Retrieval-Augmented Generation (RAG) can significantly enhance document retrieval processes. By enabling AI-driven comprehension of meaning and context behind user queries, the research focuses on providing faster, more accurate retrieval while simultaneously reducing the cognitive load on users.

Through these innovations, the thesis aims to bridge the gap between traditional document management systems and modern, AI-enhanced solutions, contributing to more efficient workflows and improved information accessibility in corporate environments.

1.2 Goals

The research aims to address the challenges of traditional document management systems by integrating advanced AI methodologies. The goals are structured as follows:

1. **Efficient Text Extraction and Structuring**
   Investigate and implement the most effective techniques for extracting and structuring text from unstructured corporate documents (e.g., PDFs) to prepare them for semantic processing.

2. **Embedding Model Selection and Implementation**
   Evaluate and select optimal embedding models to create high-quality semantic representations of corporate documents.

3. **Advanced Retrieval Optimization**
   Implement advanced retrieval strategies, including semantic reranking and Maximal Marginal Relevance (MMR), to improve retrieval accuracy and relevance.

4. **Testing Different Large Language Models (LLMs)**
   Explore and test multiple LLMs to generate responses, identifying the most suitable models for integration into the document management system.

5. **System Evaluation**
   Test and evaluate the performance of the enhanced document management system using metrics such as retrieval accuracy with MRR, response time, and user satisfaction. Additionally, assess the quality of generated outputs using deterministic metrics (e.g., BLEU, ROUGE) and LLM-based metrics. Compare these results against baseline methods to validate improvements.

By achieving these goals, this research aims to deliver a practical solution with efficient deployment time and low error rates. The proposed solution is designed to be scalable, maintaining acceptable query response times and resource efficiency as dataset size and concurrent user numbers grow. By enhancing document accessibility and retrieval efficiency, the system seeks to improve corporate workflows and user productivity through faster task completion, higher accuracy, reduced cognitive load, greater ease of use, and increased user satisfaction.

1.3 Thesis Structure

The thesis is organized into six chapters, as outlined below:

* **Chapter 1: Introduction**
  This chapter provides an overview of the thesis, outlining its context, scope, objectives, and goals.

* **Chapter 2: Background**
  This chapter reviews foundational concepts, tools, and methodologies relevant to the study. It covers topics such as text extraction techniques, embedding models,

Large Language Models (LLMs), vector databases, Retrieval-Augmented Generation (RAG), and evaluation metrics like Mean Reciprocal Rank (MRR) and LLM-based metrics.

* **Chapter 3: Related Work**
  This chapter examines prior research in document management systems and the application of LLMs in generative tasks. It identifies gaps in the existing literature and positions the thesis as addressing these gaps through innovative approaches.

* **Chapter 4: Methodology**
  This chapter describes the methodologies and techniques employed to achieve the thesis objectives. It details the text preprocessing pipeline, embedding model evaluation, advanced retrieval strategies (e.g., semantic reranking and MMR), vector database integration, and the exploration of LLMs for response generation. The evaluation metrics and experimental setup are also discussed.

* **Chapter 5: Results and Discussion**
  This chapter presents the findings of the research, including performance evaluations of retrieval strategies, embedding models, and LLMs. The results are analyzed with a focus on retrieval accuracy, generation quality, computational efficiency, and trade-offs between model performance and resource requirements.

* **Chapter 6: Conclusions**
  This chapter summarizes the key findings and contributions of the thesis. It also outlines potential areas for future research and improvements to the proposed solutions.

Chapter 2

Background

This chapter outlines the methodologies and tools employed in this study. It begins with the data preparation process, followed by an overview of LLMs, embedding models, and reranking models. The chapter then introduces vector databases and the RAG framework, concluding with the evaluation metrics, including MRR and generation-specific metrics, used to assess system performance.

2.1 Text Extraction Tools and Frameworks

Extracting text from PDF documents is a critical preprocessing step in many document processing workflows, such as information retrieval, document analysis, and natural language processing (NLP). PDF text extraction poses unique challenges due to the complex layout of PDFs, which may include multiple columns, embedded images, tables, and non-standard fonts. Over the years, various tools and frameworks have been developed to address these challenges, leveraging different techniques for extracting textual content efficiently and accurately. Here are some of the popular frameworks for text extraction from PDF documents.

2.1.1 PyPDF2

PyPDF2 is a Python library for working with PDF files, including text extraction, merging, splitting, and encryption handling. It provides features such as extracting text from PDF files, merging, splitting, and rotating pages, and handling encrypted PDFs with password protection. It is lightweight and easy to integrate into Python workflows, making it widely used in Python-based projects. However, it has limitations, including limited support for complex PDF layouts like tables or multi-column text, and its text extraction accuracy may be lower compared to other specialized tools.[3]

2.1.2 PDFMiner

PDFMiner is a Python library that provides more advanced features for extracting text from PDFs, including layout analysis. The tool offers features such as extracting text,

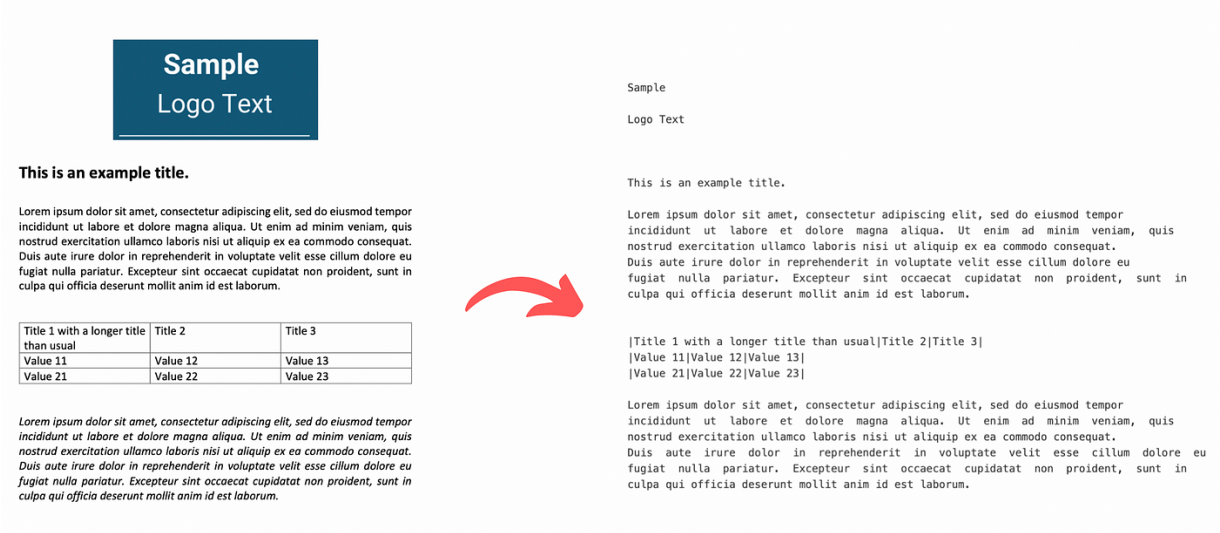


Figure 2.1: Comparison of a PDF document (left) with its extracted plain-text representation (right), highlighting the challenges in preserving structure, style, and layout during text extraction from complex documents.

metadata, and font information, with support for layout-aware extraction, enabling more accurate processing of complex documents. It provides detailed control over text extraction and layout handling, making it well-suited for extracting structured data from complex PDFs. However, it has some limitations, including slower performance compared to other libraries and a more complex setup and usage process.[4]

2.1.3 PDFPlumber

PDFPlumber is a Python library designed specifically for extracting structured data, such as tables and forms, from PDFs. It includes features like extracting text, tables, and images, along with advanced tools for handling complex layouts, such as multi-column text and nested tables. Additionally, it allows users to extract elements from specific PDF regions, providing excellent control over PDF content. Its advantages include being highly effective for extracting structured data, offering granular control over PDF elements, and easy integration with Python workflows. However, it has limitations, including slightly slower performance when handling large documents and limited support for encrypted or scanned PDFs.[5]

2.1.4 PyMuPDF (Fitz)

PyMuPDF, also known as Fitz, is a Python binding for the MuPDF library, renowned for its high efficiency in rendering and extracting content from PDFs. It offers features such as extracting text, images, and annotations, with high performance even for large documents. The library supports layout-aware text extraction and metadata handling, ensuring versatility in processing various document types. Its advantages include excellent performance and accuracy for both simple and complex PDF layouts, comprehensive documentation, active maintenance, and support for additional formats like EPUB and XPS. However, it has a slight learning curve for utilizing its advanced features effectively.[6]

2.1.5 Comparison of Text Extraction Tools

Table 2.1 highlights the strengths and limitations of popular PDF text extraction tools. PyPDF2 is simple and easy to use, making it ideal for basic tasks, but it lacks advanced layout handling and support for tables or images. PDFMiner offers advanced layout-aware extraction but is slower and less user-friendly, while PDFPlumber balances ease of use with robust support for tables and forms. PyMuPDF (Fitz) excels in speed and format versatility but has limited table extraction capabilities and moderate usability. The choice of tool ultimately depends on task requirements, with PyPDF2 suited for basic needs, PDFMiner and PDFPlumber for complex structured data, and PyMuPDF for high-performance and versatile tasks.

Feature	**PyPDF2**	**PDFMiner**	**PDFPlumber**	**PyMuPDF (Fitz)**
Text Extraction	Basic	Advanced (Layout-aware)	Advanced (Tables, Forms)	High Performance (Layout-aware)
Table Extraction	No	Limited	Yes	Limited
Image Extraction	No	No	Yes	Yes
Metadata Extraction	No	Yes	Limited	Yes
Performance	Moderate	Slower	Moderate	Fast
Ease of Use	Easy	Complex	Easy	Moderate
Layout Handling	Limited	Advanced	Advanced	Advanced
Support for Encrypted PDFs	Yes	Yes	Limited	Yes
Formats Supported	PDF Only	PDF Only	PDF Only	PDF, EPUB, XPS

Table 2.1: Comparison of Popular Text Extraction Tools for PDFs

2.2 Embedding Models

Embedding models are machine learning models designed to represent linguistic units—such as words, phrases, sentences, or documents—as dense numerical vectors in a continuous vector space. These embeddings encode semantic relationships, allowing for efficient comparison and processing of text. The concept of embeddings stems from the idea of mapping discrete textual data into a numerical format that machine learning models can effectively process [7]. Embeddings have become a cornerstone of modern natural language processing (NLP), enabling a wide range of applications, from semantic search to machine translation.

2.2.1 Types of Embedding Models

Embedding models use various architectures to generate vector representations, ranging from traditional shallow models to modern transformer-based architectures. At a high level, embedding models can be categorized into the following types:

Static Embedding Models

Static embedding models generate fixed vector representations for each word, regardless of its context in a sentence. These embeddings are precomputed and remain constant across different tasks.

* **Word2Vec**: A shallow neural network-based model that learns static embeddings by predicting either the context of a word (*Skip-Gram*) or the word itself from its context (*CBOW*) [7].

$$P(w_t \mid w_{context}) = \frac{\exp(v_{w_t} \cdot v_{context})}{\sum_{i \in V} \exp(v_i \cdot v_{context})}$$

  Here, w_t is the target word, $w_{context}$ represents the surrounding words, and v denotes the vector representation.

* **GloVe**: A model that generates embeddings by applying matrix factorization on the word co-occurrence matrix, capturing both local and global word relationships [8].

$$f(X_{ij})(w_i \cdot w_j + b_i + b_j) = \log(X_{ij})$$

  Where X_{ij} represents the word co-occurrence statistics, and w_i, w_j are the embeddings.

Subword-Based Models

Subword-based models generate embeddings by leveraging character-level features, such as n-grams, to handle out-of-vocabulary (OOV) words effectively.

* **FastText**: Builds word embeddings by combining subword (character n-gram) representations. This approach captures morphological information, making it robust for rare and unseen words [9].

$$v_w = \sum_{g \in G_w} v_g$$

  Where v_w is the word vector, G_w represents the set of n-grams for the word w, and v_g are the embeddings of the n-grams.

Contextual Embedding Models

Contextual embedding models generate dynamic vector representations for words or sentences, where the embeddings vary depending on the surrounding context. These models leverage deep learning architectures like transformers for richer semantic representations.

* **BERT**: A transformer-based model that generates contextual embeddings by training on tasks like Masked Language Modeling (MLM) and Next Sentence Prediction (NSP) [10].

$$P(w_t \mid C) = \frac{\exp(h_t \cdot v_{w_t})}{\sum_{i \in V} \exp(h_t \cdot v_i)}$$

  Here, h_t represents the hidden state of the token t, and v_{w_t} is its embedding.

* **SBERT**: An extension of BERT, SBERT uses a Siamese architecture to produce sentence-level embeddings, making it suitable for tasks like semantic similarity and clustering [11].

Table 2.2 summarizes the key differences between static, subword-based, and contextual embedding models.

Model Type	**Key Feature**	**Examples**
Static Embedding Models	Fixed embeddings, context-independent	Word2Vec, GloVe
Subword-Based Models	Handles OOV words using character-level features	FastText
Contextual Embedding Models	Dynamic embeddings, context-sensitive	BERT, SBERT

Table 2.2: Comparison of Different Types of Embedding Models

2.2.2 Key Features of Embedding Models

Key features of embedding models are:

Semantic Representation: Embedding models map text into a high-dimensional vector space where semantically similar entities are closer together. For example:

* Models like **Word2Vec** capture syntactic and semantic relationships (e.g., king - man + woman = queen).

* Contextual models like **BERT** produce embeddings that vary based on the text's context.

Context Sensitivity:

* **Static Models:** Word2Vec and GloVe assign fixed vectors to words regardless of their context.

* **Contextual Models:** Models like BERT generate embeddings that adapt based on the surrounding text, enabling richer and more flexible representations.

Multilingual Support: Modern models like FastText and multilingual versions of BERT provide support for multiple languages, enabling cross-lingual tasks like machine translation and multilingual search.

2.2.3 Embedding Models Used in This Project

This project evaluated the following state-of-the-art embedding models to generate dense vector representations for semantic search and information retrieval tasks. The models used in this project are:

* **mxbai-embed-large**: A transformer-based model optimized for generating high-dimensional embeddings for complex semantic queries [12]. It is designed to handle advanced semantic search tasks, providing embeddings that are highly detailed and nuanced. The model effectively captures complex textual relationships, making it well-suited for large-scale retrieval systems. It is optimized for high-dimensional representation, ensuring fine-grained semantic understanding.

  The model's key strengths include its ability to provide semantic representations that capture intricate text relationships and maintain contextual awareness efficiently through its transformer-based architecture. While primarily focused on monolingual tasks, it offers exceptional performance in semantic search scenarios.

* **nomic-embed-text**: A multilingual embedding model designed for cross-lingual tasks [13]. It provides consistent and high-quality embeddings across multiple languages, excelling in cross-lingual semantic similarity and making it ideal for multilingual datasets. The model is robust in handling diverse linguistic structures within a unified embedding space.

  The model's key strengths include its semantic representation, optimized for multilingual consistency, and its ability to generate contextual embeddings by adapting to surrounding text. It demonstrates robust performance in both multilingual and cross-lingual tasks, making it a powerful tool for diverse linguistic applications.

* **bge-m3-f16**: A lightweight embedding model that balances computational efficiency and accuracy [14]. It is specifically designed for low-latency applications, ensuring quick responses in retrieval systems. The model prioritizes computational efficiency without compromising the quality of semantic representation, making it effective for real-time semantic search in resource-constrained environments.

  The model's key strengths include robust semantic representation, maintaining semantic accuracy while emphasizing efficiency. It efficiently handles contextual variations using a lightweight transformer architecture. Although primarily focused on monolingual tasks, it provides highly efficient embeddings for semantic search, catering to real-time applications.

* **multilingual-e5-large**: A transformer-based model fine-tuned for multilingual tasks [15]. It captures detailed semantic relationships across multiple languages, delivering robust performance in cross-lingual retrieval and multilingual information systems. The model excels in scenarios that require accurate representation of diverse linguistic contexts.

  The model's key strengths include its ability to capture intricate semantic relationships in multilingual settings, its context sensitivity through dynamically adaptive embeddings, and its effectiveness in cross-lingual tasks and multilingual datasets. These features make it an excellent choice for handling complex multilingual applications.

Table 2.3 summarizes the key features of the embedding models used in this project.

Model	Architecture	Multilingual Support	Specialization
mxbai-embed-large	Transformer-based	No	High-dimensional embeddings for complex semantic queries
nomic-embed-text	Transformer-based	Yes	Cross-lingual semantic similarity
bge-m3-f16	Lightweight Transformer	No	Low-latency embeddings with computational efficiency
multilingual-e5-large	Transformer-based	Yes	Semantic embeddings for multilingual tasks

Table 2.3: Comparison of Embedding Models Used in This Project

2.3 Reranking Models

Reranking models refine the order of candidates retrieved by an initial retriever, improving relevance through deeper semantic, contextual, or task-specific analysis. These models are widely used in applications such as search engines, recommendation systems, and question answering, where precise ranking is critical to user satisfaction [16].

By leveraging additional computational resources, reranking models enhance ranking quality by incorporating detailed contextual insights. They are highly adaptable to specific domains and tasks, making them versatile for various use cases. However, their increased computational demands can affect real-time systems, and their performance relies heavily on high-quality annotated training data [17].

2.3.1 Notation

Given a query q and an initial ranked list of documents $D = \{d_1, d_2, \ldots, d_k\}$, a reranking model assigns a new relevance score $R(q, d_i)$ to each document d_i in D to produce a reranked list. The reranking function is expressed as:

$$R(q, d_i) = f(q, d_i; \theta)$$

where:

* f: The reranking function parameterized by θ,
* q: The query,
* d_i: The candidate document,
* $R(q, d_i)$: The new relevance score.

The reranked list is computed as:

$$D_{reranked} = \text{sort}(D, R(q, d_i)).$$

The objective of a reranker is to maximize ranking performance metrics such as NDCG (Normalized Discounted Cumulative Gain) [18] and MAP (Mean Average Precision)[17]. These metrics are defined as follows:

$$\text{NDCG@k} = \frac{1}{Z_k} \sum_{i=1}^{k} \frac{2^{\text{rel}_i} - 1}{\log_2(i+1)} \tag{2.1}$$

Here, rel_i represents the relevance of the result at rank i, and Z_k is a normalization factor to ensure the score is between 0 and 1.

$$\text{MAP} = \frac{1}{Q} \sum_{q=1}^{Q} \frac{1}{R_q} \sum_{k=1}^{R_q} P(k) \tag{2.2}$$

In the MAP formula, Q is the total number of queries, R_q is the number of relevant documents for query q, and $P(k)$ is the precision at rank k.

The reranker optimizes these metrics to improve the quality of ranked results, ensuring that relevant documents are ranked higher in the retrieval list.

2.3.2 Types of Reranker Models

Reranking models can be categorized based on their architectures: **dense retrievers**, **cross-encoders**, and **sequence generation models**. Each architecture has unique strengths and trade-offs, making them suitable for different tasks.

Dense Retriever-Based Models

Dense retrievers encode queries and documents independently, generating dense vector representations that are compared for relevance.

1. Sentence-BERT (SBERT) Rerankers SBERT generates dense vector embeddings for queries and documents independently and calculates their relevance using cosine similarity:

$$\cos(E_q, E_{d_i}) = \frac{E_q \cdot E_{d_i}}{\|E_q\| \|E_{d_i}\|}$$

where:

* $E_q \cdot E_{d_i}$: The dot product of the query embedding E_q and the document embedding E_{d_i}, computed as:

$$E_q \cdot E_{d_i} = \sum_{j=1}^{n} E_{qj} \cdot E_{d_{ij}}$$

* $\|E_q\|$: The magnitude (Euclidean norm) of the query embedding E_q, given by:

$$\|E_q\| = \sqrt{\sum_{j=1}^{n} E_{qj}^2}$$

* $\|E_{d_i}\|$: The magnitude (Euclidean norm) of the document embedding E_{d_i}, given by:

$$\|E_{d_i}\| = \sqrt{\sum_{j=1}^{n} E_{d_{ij}^2}}$$

SBERT-based rerankers are computationally efficient since embeddings are precomputed, making them suitable for large-scale semantic search tasks [11].

2. BGE-Reranker-v2-m3 The BGE-Reranker-v2-m3 is a transformer-based reranking model that builds upon dense embeddings to enhance retrieval tasks. It leverages pre-trained embeddings and fine-tuned contextual interactions for robust semantic search [19]. Key features include:

* High precision for semantic search applications.
* Optimized performance for multi-domain ranking tasks.
* Balances computational efficiency and ranking accuracy.

This model was used in this project to rerank candidates in the retrieval pipeline, significantly improving relevance and precision.

Cross-Encoder Models

Cross-encoders jointly encode both the query and candidate document, allowing token-level interactions to be modeled for precise relevance scoring.

1. BERT-Based Rerankers BERT-based models calculate the relevance score by jointly encoding the query q and document d_i using a transformer architecture with cross-attention mechanisms to capture token-level interactions. The relevance score is computed as:

$$R(q, d_i) = \text{softmax}(W \cdot h_{[CLS]}) = \frac{\exp(W \cdot h_{[CLS]} + b)}{\sum_{j=1}^{N} \exp(W \cdot h_{[CLS]}^{(j)} + b)},$$

where:

* $h_{[CLS]}$: The representation of the `[CLS]` token from the final layer of the BERT model, capturing the overall interaction between q and d_i,
* W: A learned weight matrix applied to the `[CLS]` embedding,
* b: A bias term.

Notable examples of BERT-based rerankers include BERT for passage ranking [20] and MonoBERT [21]. These models excel in tasks requiring fine-grained contextual understanding by leveraging cross-attention but are computationally intensive due to their reliance on joint encoding of queries and documents.

2. ColBERT ColBERT (Late Interaction Architecture) computes token-level relevance between query and document representations. It strikes a balance between efficiency and precision by independently encoding query and document tokens and performing late interaction for relevance scoring [22].

Sequence Generation Models

These models approach reranking as a sequence generation task, formulating relevance scoring as a language modeling problem.

1. MonoT5 MonoT5 reformulates reranking as a sequence generation task, where the model generates tokens like "relevant" or "not relevant" as relevance scores:

$$R(q, d_i) = P(\text{"relevant"} \mid q, d_i)$$

This approach, introduced in [23], demonstrates the effectiveness of generative models for reranking tasks. MonoT5 can handle complex ranking scenarios by incorporating generative modeling capabilities.

2.3.3 Advantages and Challenges of Reranking Models

* **Advantages**:
  - Enhanced precision through detailed contextual analysis.
  - Adaptable to specific tasks or domains.
  - Improves downstream performance for search, recommendation, and QA tasks.
* **Challenges**:
  - Increased computational cost compared to initial rankers.
  - Dependency on high-quality training data for optimal performance.
  - Potential latency issues in real-time systems.

2.4 Large Language Models (LLMs)

Large Language Models (LLMs) are advanced natural language processing (NLP) systems designed to understand, generate, and manipulate human language. Their capabilities arise from extensive training on diverse datasets, enabling them to perform tasks such as summarization, question answering, and text generation. Most LLMs utilize the transformer architecture, introduced by Vaswani et al. (2017) [24], as their foundational framework.

2.4.1 Key Features and Architecture

Transformer Architecture

Most LLMs such as GPT, BERT and LLaMA are built on the **transformer model**, which employs self-attention to capture relationships between tokens in a sequence:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

where Q, K, and V are the query, key, and value matrices, and d_k is the dimensionality of the key vectors. Positional encoding is added to incorporate sequential information, enabling the model to capture global dependencies in text [24].

Context Window

The **context window** determines the number of tokens a model can process in a single instance. Recent LLMs support increasingly large context windows:

* **GPT-4:** Up to 32,768 tokens [25].
* **LLaMA 3:** Typically 8,000 tokens, extendable in advanced versions [26].
* **Mixtral:** Up to 64,000 tokens for processing extensive documents [27].

Larger context windows enable tasks like summarizing long documents and complex reasoning.

Parameter Size

The size of an LLM, defined by the number of parameters, determines its capacity:

* **Small models:** 1–6 billion parameters (e.g., GPT-2).
* **Medium models:** 6–30 billion parameters (e.g., GPT-3, LLaMA 2 7B).
* **Large models:** 70+ billion parameters (e.g., GPT-4, LLaMA 3 70B) [26].

Pooling Mechanisms

Pooling mechanisms are used to aggregate token embeddings into a fixed-size representation of the input sequence. This aggregation step is crucial in downstream tasks like sentiment classification and topic modeling, where the model requires a sequence-level representation. The paper [28] discusses three primary pooling mechanisms: average pooling, max pooling, and attentive pooling. Additionally, CLS pooling is commonly employed in transformer-based models for classification tasks.

* **Average Pooling**
  Average pooling computes the mean of all token embeddings across the sequence. It treats all tokens equally, providing a smooth representation of the overall input. While simple and effective for many tasks, it may fail to emphasize important words in the sequence. As illustrated in Figure 2.2, average pooling aggregates all tokens equally, resulting in a balanced but generalized representation.

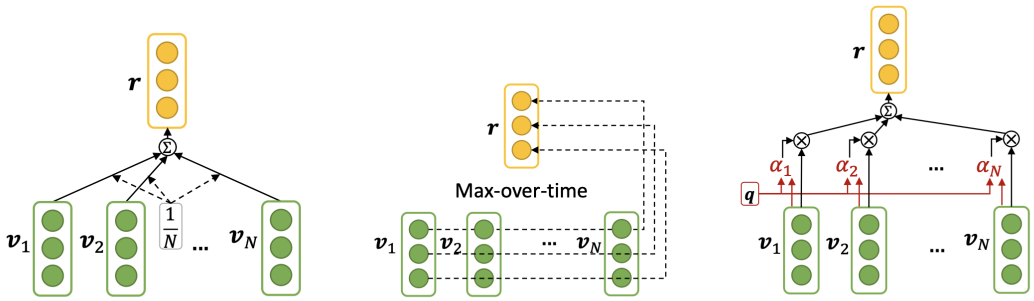
* **Max Pooling**
  Max pooling selects the maximum value for each dimension across all token embeddings. This method highlights the most significant features within the input sequence, focusing on the most salient words. Max pooling can be useful when specific tokens (e.g., highly activated embeddings) are key for the task at hand. Figure 2.2 demonstrates how max pooling captures the most prominent features of the input.

* **Attentive Pooling**
  Attentive pooling assigns different weights to tokens based on their importance to the task. The attention mechanism computes weights $(\alpha_1, \alpha_2, \ldots, \alpha_N)$ for each token, enabling the model to focus on task-relevant words. As shown in Figure 2.2, attentive pooling generates a weighted sum of token embeddings, providing a more task-specific representation. This mechanism is particularly effective for tasks like sentiment classification, where specific words (e.g., "good") carry more significance.

* **CLS Pooling**
  CLS pooling is used in transformer-based architectures like BERT for tasks requiring a single embedding representation of the entire sequence. The `[CLS]` token is a special token prepended to the input sequence, and its corresponding embedding from the final transformer layer is directly used for downstream tasks (e.g., classification). Unlike other pooling mechanisms that aggregate embeddings, CLS pooling leverages the transformer's self-attention mechanism to encode global context into the `[CLS]` embedding.

$$\text{CLS Representation} = \text{Final Transformer Layer Output}(\texttt{[CLS]})$$

CLS pooling is particularly effective for tasks where sequence-level representations are required, such as document classification and sentiment analysis.



(a) Average pooling. (b) Max pooling. (c) Attentive pooling.

Figure 2.2: Illustration of pooling mechanisms: (a) Average pooling, which computes the mean of token embeddings; (b) Max pooling, which selects the maximum value per dimension; (c) Attentive pooling, which weights token embeddings using attention scores; and (d) CLS pooling, which uses the embedding of the `[CLS]` token. Adapted from [28].

Task-Specific Pooling Comparison Figure 2.3 compares the performance of these pooling mechanisms in two tasks: sentiment classification and news topic classification. Average pooling provides a balanced overview of the sequence, max pooling emphasizes the most activated embeddings, attentive pooling selectively weights task-relevant tokens, and CLS pooling leverages the transformer's self-attention mechanism to generate a global sequence representation.

Sentiment Classification	
Average Pooling	The movie is good, but not to my taste
Max Pooling	The movie is good, but not to my taste
Attentive Pooling	The movie is good, but not to my taste
News Topic Classification	
Average Pooling	Fire on Queensland Island Takes Heavy Toll on Wildlife
Max Pooling	Fire on Queensland Island Takes Heavy Toll on Wildlife
Attentive Pooling	Fire on Queensland Island Takes Heavy Toll on Wildlife

Figure 2.3: Task-specific comparison of pooling mechanisms. Highlighted regions indicate the focus of each pooling method for sentiment classification and news topic classification tasks. Attentive pooling and CLS pooling consistently assign higher importance to task-relevant tokens. Adapted from [28].

2.4.2 LLMs Used in This Project

This project evaluated several state-of-the-art LLMs to identify the most effective models for specific NLP tasks, such as information retrieval and document summarization. Below is an overview of the LLMs tested:

* **Mixtral 8x22B Instruct:** A Sparse Mixture of Experts (SMoE) model with 141 billion parameters (39 billion active during inference). It supports a 64,000-token context window and excels in multilingual tasks and coding, released under the Apache 2.0 license [29].

* **Mixtral 8x7B:** A smaller SMoE model with 46.7 billion parameters (12.9 billion active). It features a 32,000-token context window and performs well in code generation and multilingual tasks, also Apache 2.0 licensed [30].

* **Mistral-B-v0.3:** A 123-billion-parameter model with a 128,000-token context window. It demonstrates strong performance in programming tasks and supports multiple languages [31].

* **LLaMA 3 8B Instruct FP16:** An 8-billion-parameter model with an 8,000-token context window. It is fine-tuned on 10 million human-annotated examples and achieves strong performance in NLP benchmarks like MMLU [32].

* **LLaMA 3 70B Instruct Q4_0:** A 70-billion-parameter quantized model supporting an 8,000-token context window, fine-tuned on extensive human-labeled data for high accuracy and efficiency [33].

* **Gemma2:9b:** A smaller, efficient model used for baseline comparisons. Details on its performance are limited due to proprietary constraints [34].

2.4.3 Comparison of LLMs

Table 2.4 compares the key features of the evaluated models, highlighting their context window size, parameter count, and areas of specialization.

Model	**Parameters (Billion)**	**Context Window (Tokens)**	**Specialization**
Mixtral 8x22B Instruct	141 (39 active)	64,000	Multilingual tasks, coding
Mixtral 8x7B	46.7 (12.9 active)	32,000	Multilingual tasks, code generation
Mistral-B-v0.3	123	128,000	Programming tasks, multilingual support
LLaMA 3 8B Instruct FP16	8	8,000	Instruction-following, NLP benchmarks
LLaMA 3 70B Instruct Q4_0	70	8,000	High accuracy, quantized efficiency
Gemma2:9b	9	Variable	General-purpose tasks

Table 2.4: Comparison of Large Language Models Used in This Project

2.5 Vector Databases (VectorDB)

Vector databases (VectorDBs) are specialized databases designed to store, index, and query high-dimensional vector representations of data. These vectors, often generated by machine learning models, encode semantic information about text, images, audio, or other data types. VectorDBs have become integral to modern applications such as semantic search, recommendation systems, anomaly detection, and AI-driven knowledge retrieval.

2.5.1 Key Features of Vector Databases

* **Efficient Similarity Search**
  VectorDBs optimize similarity-based operations, such as k-nearest neighbor (k-NN) search, enabling fast and scalable retrieval of items that are semantically similar to a query vector. Algorithms such as Approximate Nearest Neighbor (ANN) are often used for computational efficiency [35].

* **Indexing and Scalability**
  Advanced indexing techniques, including Hierarchical Navigable Small World (HNSW),

Inverted File (IVF), and Product Quantization (PQ), allow VectorDBs to handle billions of vectors effectively [36].

* **Integration with Machine Learning**
  VectorDBs seamlessly integrate with embeddings generated by pre-trained models like BERT, OpenAI embeddings, or custom neural networks. They also support multimodal vectors, enabling the storage of representations for text, images, audio, and video data.

* **Real-Time Updates**
  Many VectorDBs allow dynamic updates, such as adding or removing vectors, without the need to re-index the entire dataset, making them suitable for real-time applications.

2.5.2 ChromaDB: A Specialized Vector Database

ChromaDB is an open-source vector database optimized for AI-driven applications, particularly in workflows involving large language models (LLMs). It emphasizes simplicity, high performance, and seamless integration into modern machine learning pipelines. Designed with a focus on embedding storage and querying, ChromaDB has emerged as a key component in Retrieval-Augmented Generation (RAG) systems.

Key Features of ChromaDB

* **Efficient Embedding Storage:**
  ChromaDB stores high-dimensional embeddings alongside metadata, enabling complex structured queries that combine semantic similarity and metadata filtering.

* **Advanced Semantic Querying:**
  It supports similarity search using metrics such as cosine similarity and Euclidean distance, ensuring fast and accurate retrieval of relevant embeddings.

* **Metadata-Driven Search:**
  ChromaDB enhances retrieval by allowing queries to filter based on metadata (e.g., categories, timestamps, or user preferences), making it suitable for personalized search applications.

* **Seamless Integration:**
  ChromaDB is designed to integrate effortlessly with popular Python frameworks and libraries, including LangChain and FastAPI, enabling quick adoption by developers and researchers.

* **Optimization for RAG Pipelines:**
  Specifically tailored for Retrieval-Augmented Generation workflows, ChromaDB enables efficient storage and retrieval of embeddings to provide context for LLM-based applications.

* **Open Source and Community-Driven:**
  As an actively maintained project, ChromaDB benefits from a growing community that continuously contributes to its features and integrations [37].

Comparison: General VectorDBs vs. ChromaDB

Table 2.5 compares general-purpose vector databases (VectorDBs) with ChromaDB, highlighting their key differences. General VectorDBs are versatile and cater to a broad range of similarity search tasks, offering wide API support and indexing algorithms like IVF, HNSW, and PQ. In contrast, ChromaDB is tailored for AI and machine learning applications, with a Python-centric design and integration with LangChain, making it ideal for semantic search, Retrieval-Augmented Generation (RAG), and AI pipelines. Additionally, ChromaDB provides advanced, user-friendly metadata querying, distinguishing it from other general-purpose VectorDBs.

Feature	**General VectorDBs**	**ChromaDB**
Primary Focus	General-purpose	AI and machine learning applications
Integration	Wide API support	Python-centric with LangChain support
Indexing Algorithms	IVF, HNSW, PQ	Optimized for fast similarity search
Metadata Querying	Varies	Advanced and user-friendly
Use Cases	General similarity search	Semantic search, RAG, AI pipelines

Table 2.5: Comparison of General VectorDBs and ChromaDB

2.6 Retrieval-Augmented Generation (RAG): An Overview

Retrieval-Augmented Generation (RAG) is an advanced natural language processing (NLP) framework designed to address knowledge-intensive tasks by combining document retrieval and text generation. Introduced by [1], RAG integrates two primary components: a **retriever**, which identifies relevant documents or passages from a predefined knowledge base, and a **generator**, which synthesizes responses using the retrieved content. This architecture enables RAG to dynamically incorporate external knowledge, offering capabilities that go beyond pre-trained large language models (LLMs).

Unlike traditional LLMs that rely solely on static pre-trained parameters, RAG leverages a retriever to fetch the most relevant and up-to-date information from an external source, such as a vector database or document index. This decoupled architecture allows the retriever and generator to function independently, enabling greater flexibility and scalability. RAG is particularly effective in scenarios requiring real-time access to domain-specific or dynamic knowledge.

2.6.1 How RAG Works

The RAG process is divided into two main stages: indexing and retrieval.

Indexing Stage

In the indexing stage, documents are pre-processed to enable efficient retrieval. This involves:

* Cleaning the documents and enriching them with metadata.
* Splitting the documents into smaller, manageable segments (also known as chunking).
* Embedding the segments using a pre-trained embedding model.
* Storing the embeddings in a vector database for efficient similarity-based retrieval.

This process is typically performed offline, ensuring the system is ready for efficient online retrieval. Figure 2.4 illustrates the indexing pipeline.

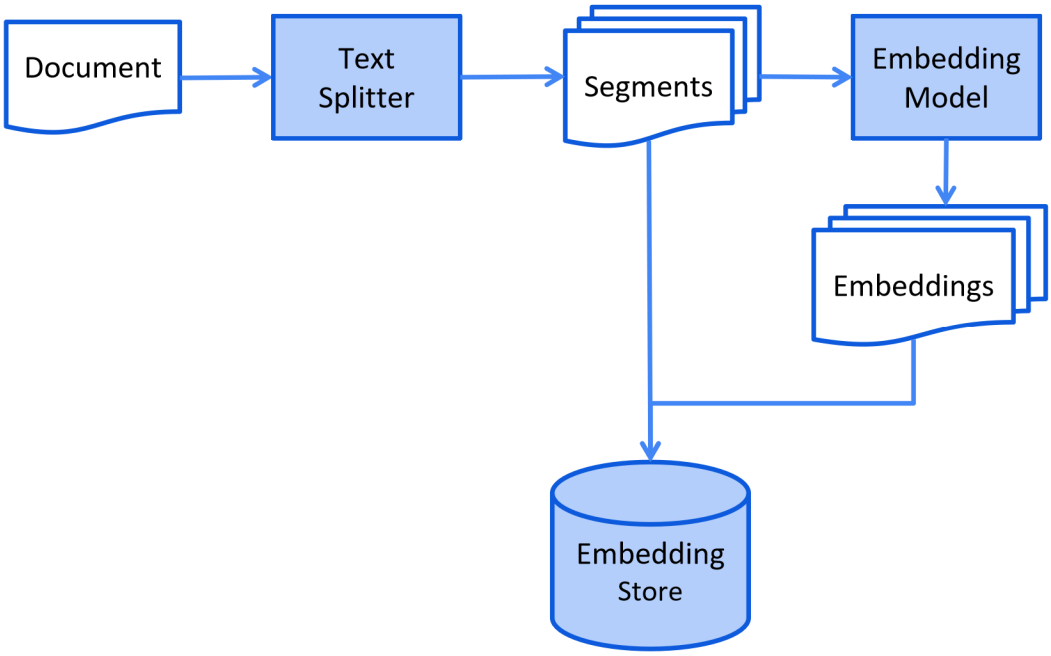


Figure 2.4: The indexing stage in RAG: Documents are split into segments, embedded using an embedding model, and stored in a vector database for efficient retrieval.

Retrieval Stage

During the retrieval stage, the user submits a query, which is embedded using the same embedding model as in the indexing stage. The retriever identifies the most relevant segments from the vector database using similarity search (e.g., cosine similarity), providing context to the generator. Figure 2.5 shows a simplified retrieval pipeline.

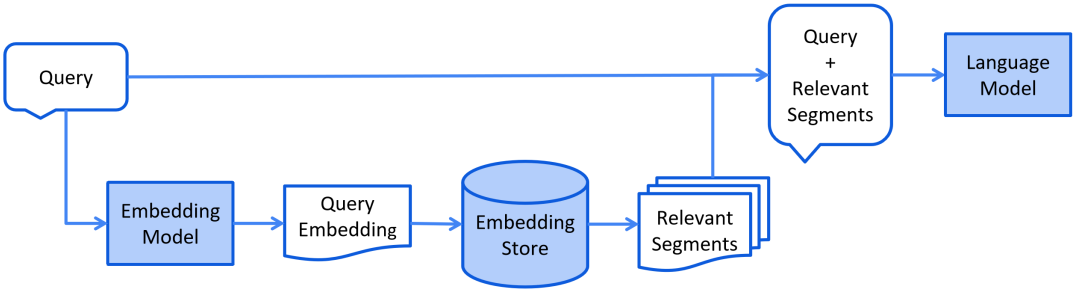


Figure 2.5: The retrieval stage in RAG: The query is embedded, and relevant segments are retrieved to generate context-aware responses.

Retriever Model

The retriever component in Retrieval-Augmented Generation (RAG) identifies the most relevant documents or passages based on the semantic similarity between the query and document embeddings. The similarity is computed using metrics such as cosine similarity:

$$\text{Similarity}(q, d_i) = \cos(E_q, E_{d_i}) = \frac{E_q \cdot E_{d_i}}{\|E_q\| \|E_{d_i}\|}$$

where:

* q: The query,
* d_i: A document or passage,
* E_q and E_{d_i}: The embeddings of the query and document,
* $E_q \cdot E_{d_i}$: The dot product of the query and document embeddings,
* $\|E_q\|$: The magnitude (Euclidean norm) of the query embedding, calculated as:

$$\|E_q\| = \sqrt{\sum_{k=1}^{n} E_{q_k}^2},$$

* $\|E_{d_i}\|$: The magnitude (Euclidean norm) of the document embedding, calculated similarly, and
* cos: The cosine similarity metric.

This formula calculates the cosine of the angle between two vectors E_q and E_{d_i}, where:

* A value close to 1 indicates high similarity,
* A value close to 0 indicates orthogonality (no similarity),
* A value close to -1 indicates high dissimilarity.

Popular retriever models include Dense Passage Retriever (DPR) [38] and BERT-based retrievers, which leverage dense embeddings to enhance retrieval accuracy for semantic search tasks.

Generator Model

The generator takes the retrieved documents along with the original query to produce a coherent and contextually appropriate response. This is typically implemented using autoregressive models such as GPT. The conditional probability of generating the answer is expressed as:

$$P(a \mid q, \{d_1, d_2, \ldots, d_k\}) = \prod_{t=1}^{T} P(a_t \mid a_{<t}, q, \{d_1, d_2, \ldots, d_k\})$$

where:

* a is the generated answer,
* T is the length of the generated answer,
* $\{d_1, d_2, \ldots, d_k\}$ are the top k retrieved documents.

2.6.2 Variants of RAG

Two key variants of RAG are proposed by [1]:

* **RAG-Token:** This approach marginalizes over the retrieved documents at the token level. The probability of generating an output token is computed as:

  $$P(a \mid q) = \sum_{d \in D} P(a \mid q, d) P(d \mid q)$$

  This method allows fine-grained consideration of multiple documents during token generation.

* **RAG-Sequence:** This variant marginalizes over entire document sequences, treating each retrieved document independently during the generation process:

  $$P(a \mid q) = \prod_{t=1}^{T} \sum_{d \in D} P(a_t \mid a_{<t}, q, d) P(d \mid q)$$

  It simplifies computation but may lose inter-document coherence.

2.6.3 Applications of RAG

RAG has shown significant potential in:

* **Question Answering:** Providing precise answers to complex user queries by retrieving relevant context and generating accurate responses.
* **Summarization:** Synthesizing concise summaries of large documents or datasets.
* **Knowledge Retrieval:** Enabling dynamic, real-time access to domain-specific information repositories.

By combining the precision of retrieval with the generative power of language models, RAG has become a powerful framework for tasks requiring real-time, context-aware knowledge integration.

2.7 Evaluation Metrics for RAG Systems

The evaluation of the RAG pipeline assessed the quality of retrieved documents using **MRR** and the generated responses through metrics divided into two categories: *deterministic metrics*, including **Deterministic Correctness** and **Deterministic Faithfulness**, and *LLM-based metrics*, such as **LLM-based correctness**, **LLM-based faithfulness** and **LLM-based style consistency**.

2.7.1 Mean Reciprocal Rank (MRR)

Mean Reciprocal Rank (MRR) [39] is a widely used evaluation metric in information retrieval and question answering systems to measure the effectiveness of ranked results. It is defined as:

$$MRR = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{\text{rank}_i}$$

where:

* $|Q|$: The total number of queries in the evaluation set.
* rank_i: The rank position of the first correct or relevant document for the i-th query.

The MRR metric provides an average of the reciprocal ranks over all queries. A higher MRR value (closer to 1) indicates better performance, as it implies that relevant documents are ranked higher in the results.

MRR is particularly useful for tasks where it is essential to retrieve relevant results at the top of the ranked list, such as search engines or recommender systems. However, it only considers the first relevant result and does not account for additional relevant documents lower in the ranking.

Deterministic Correctness

The **Correctness metric** evaluates how closely a generated answer matches the ground truth reference answers. It utilizes a combination of deterministic metrics to assess the relationship between the generated answer and one or more ground truth answers. The required inputs are:

* **answer:** The generated answer.
* **ground_truths:** A list of reference ground truth answers.

The Correctness metric incorporates several individual metrics for evaluation:

1. **ROUGE-L:** Measures the **longest common subsequence (LCS)** between the generated answer and the ground truth answers. This metric evaluates:
    * **Precision:** The proportion of the LCS in the generated answer compared to the ground truth.
    * **Recall:** The proportion of the LCS in the ground truth compared to the generated answer.

    * **F1-score:** The harmonic mean of Precision and Recall, providing a balanced measure of both.

2. **Token Overlap:** Computes the token-level overlap between the generated answer and the ground truth answers. This metric includes:

    * **Precision:** The fraction of tokens in the generated answer that match the ground truth.
    * **Recall:** The fraction of tokens in the ground truth that match the generated answer.
    * **F1-score:** The harmonic mean of Precision and Recall, providing a balanced measure of both.

3. **BLEU (Bilingual Evaluation Understudy):** Measures the n-gram precision between the generated answer and the ground truth answers. The BLEU score is calculated as:

$$\text{BLEU} = \text{BP} \cdot \exp\left(\sum_{n=1}^{N} w_n \log p_n\right)$$

    Where:

    * p_n is the n-gram precision, which measures the overlap of n-grams between the generated answer and the ground truth.
    * w_n is the weight assigned to each n-gram.
    * BP is the brevity penalty, which penalizes short answers that do not adequately capture the ground truth.

The score for each metric ranges between **0 and 1**, where:

* **0** indicates no similarity or overlap with the ground truth.
* **1** indicates a perfect match with the ground truth.

When multiple ground truth reference answers are provided, the maximum score across all ground truth answers is selected.

Deterministic Faithfulness

The **Faithfulness metric** evaluates how well the generated answer is grounded in the retrieved contexts. It measures the extent to which the information in the generated answer can be supported or justified by the retrieved contexts. The required inputs are:

* **retrieved_context:** A list of retrieved context sentences used to generate the answer.
* **answer:** The generated answer.

Faithfulness combines multiple deterministic metrics for evaluation:

1. **ROUGE-L Precision:** Measures the **longest common subsequence (LCS)** between the generated answer and the retrieved contexts. This metric evaluates:
   * **Precision:** The proportion of the LCS in the generated answer compared to the retrieved contexts.

   $$\text{ROUGE-L Precision} = \frac{\text{Longest Common Subsequence (LCS) between Answer and Contexts}}{\text{Length of Generated Answer}}$$

2. **Token Overlap Precision:** Calculates the precision of token-level overlap between the generated answer and the retrieved contexts. This metric evaluates:
   * **Precision:** The fraction of tokens in the generated answer that match the retrieved contexts.

   $$\text{Token Overlap Precision} = \frac{\text{Count of Common Tokens between Answer and Contexts}}{\text{Total Tokens in Generated Answer}}$$

3. **BLEU (Bilingual Evaluation Understudy):** Measures the n-gram precision between the generated answer and the retrieved contexts. The BLEU score is calculated as:

   $$\text{BLEU} = \text{BP} \cdot \exp\left(\sum_{n=1}^{N} w_n \log p_n\right)$$

   Where:
   * p_n is the n-gram precision, which measures the overlap of n-grams between the generated answer and the retrieved contexts.
   * w_n is the weight assigned to each n-gram.
   * BP is the brevity penalty, which penalizes short answers that do not adequately match the retrieved contexts.

4. **Rouge | Token Overlap | Bleu Faithfulness:** This is defined as the proportion of sentences in the generated answer that can be matched to the retrieved context above a threshold.

   $$\text{Rouge | Token Overlap | Bleu Faithfulness} = \frac{\text{Number of Sentences in Answer Matched to Context above Threshold}}{\text{Total Number of Sentences in Generated Answer}}$$

Faithfulness is defined as the proportion of sentences in the generated answer that can be matched to the retrieved contexts above a threshold. The threshold defaults to **0.5** for a sentence to be considered faithful.

The score for each metric ranges between **0 and 1**, where:

* **0** indicates no faithfulness or alignment with the retrieved contexts.
* **1** indicates perfect grounding of the generated answer in the retrieved contexts.

The `by_sentence` values represent sentence-level ROUGE, Token Overlap, and BLEU scores for each sentence in the answer. Sentences with scores above the threshold (default: 0.5) are considered faithful.

LLM-based Answer Correctness

The **LLM-based Answer Correctness** metric provides a score between **0.0 and 1.0**, evaluating the overall quality of a generated answer based on its relevance and correctness with respect to the given question and ground truth answers. To compute the LLM-based correctness score, the following inputs are required:

* **question:** The question posed to the system.
* **answer:** The generated answer to the question.
* **ground_truths:** A list of reference ground truth answers.

The score is determined using the following rubric, as defined in the LLM prompt:

* **0.0:** The answer is completely irrelevant to the question.
* **0.25:** The answer is relevant to the question but contains major errors.
* **0.5:** The answer is relevant to the question and is partially correct.
* **0.75:** The answer is relevant to the question and is correct.
* **1.0:** The answer is relevant to the question, fully correct, and complete.

LLM-Based Faithfulness

LLM-Based Faithfulness evaluates how well the generated answer is grounded in the retrieved contexts. It assesses whether the statements in the generated answer can be attributed to the retrieved contexts. The following data items are required for evaluation:

* **answer:** The generated answer to be evaluated.
* **retrieved_context:** A list of retrieved context sentences that the answer should be grounded upon.

There are two approaches for evaluating faithfulness using an LLM:

1. **Classify Faithfulness by Statement:** When `classify_by_statement = TRUE`, the LLM evaluates each statement in the generated answer individually, assigning a granular faithfulness score to every statement. This approach:
    * Provides more detailed insights into faithfulness.
    * Requires more tokens for evaluation, as each statement is processed separately.
    * Is generally more reliable, as it allows the LLM to reason over individual statements.

    The formula for faithfulness by statement is:

$$\text{LLM-Based Faithfulness} = \frac{\text{Number of Statements in Generated Answer Attributed to Retrieved Contexts}}{\text{Total Number of Statements in Generated Answer}}$$

2. **Classify Faithfulness by Whole Answer:** When `classify_by_statement = FALSE`, the LLM evaluates the entire generated answer as a single unit and assigns a binary score:

   * **1.0:** The generated answer is fully faithful to the retrieved contexts.
   * **0.0:** The generated answer is not faithful to the retrieved contexts.

   This approach is faster and requires fewer tokens, but it may overlook detailed issues within individual statements.

LLM-Based Style Consistency

LLM-Based Style Consistency assesses how closely the style of a generated answer aligns with the style of the reference answer(s). The metric produces a score between **0.0 and 1.0**, evaluating stylistic elements such as tone, verbosity, formality, complexity, and use of terminology. The following data items are required to evaluate style consistency:

* **answer:** The generated answer to be assessed.
* **ground_truths:** A list of reference answers used for comparison.

The metric outputs both a style consistency score and reasoning for the assigned score. The score is determined according to the following rubric:

* **0.0:** The generated answer has a completely different style compared to the reference answer(s).
* **0.333:** The generated answer barely matches the style of the reference answer(s), with noticeable differences.
* **0.666:** The generated answer is largely in the same style as the reference answer(s), but slight differences are present in some aspects.
* **1.0:** There is no discernible difference in style between the generated answer and the reference answer(s).

Chapter 3

Related Work

In this chapter, we review the most relevant works related to the subject of this thesis. These works address similar problems or propose methodologies and frameworks that can be adapted to the challenges faced in this research.

3.1 Traditional Approaches to Document Management Systems

Before the advent of artificial intelligence (AI), document management systems (DMS) relied on traditional methods for information retrieval. These methods, while foundational, had several limitations that modern AI-driven approaches aim to address.

3.1.1 Keyword-Based Search

Keyword-based search relied on exact string matching to retrieve documents based on user queries. This method was computationally efficient but lacked semantic understanding, often resulting in irrelevant or incomplete results. For example, a query for "report" would not retrieve documents containing synonyms like "summary" or "analysis" unless explicitly mentioned [40].

3.1.2 Boolean Search

Boolean search introduced logical operators like AND, OR, and NOT to refine queries, providing more control over search results. Despite its precision in specific scenarios, Boolean search required users to understand complex query syntax and was still limited to exact term matching, missing semantic nuances [41].

3.1.3 Metadata-Based Search

Metadata-based search utilized structured attributes (e.g., title, author, date) for retrieval. While this approach allowed precise filtering, its effectiveness was constrained by the availability and consistency of metadata, which often required manual tagging [42].

3.1.4 Full-Text Search

Full-text search indexed the entire content of documents, enabling retrieval based on any term within the text. Although it provided better coverage than keyword or metadata-based search, it lacked semantic understanding, often resulting in information overload and irrelevant results [43].

3.1.5 Hierarchical and Faceted Browsing

Hierarchical and faceted browsing organized documents into predefined categories or facets, enabling users to refine searches dynamically. However, maintaining hierarchical structures required significant effort, and the approach was less effective for unstructured or poorly tagged data [44].

3.1.6 Rule-Based Information Retrieval

Rule-based systems applied predefined patterns to extract specific information from documents. These systems were useful in specialized domains (e.g., legal, medical) but lacked adaptability to new or unstructured data formats [45].

3.2 Document Management Systems Integrated with AI

The integration of AI into document management systems has introduced advanced techniques for efficient information retrieval, contextual understanding, and user-friendly interactions. Below are examples of AI methods used in modern document management systems:

3.2.1 Examples of AI Methods

* **Retrieval-Augmented Generation (RAG)**: Combines information retrieval with generative models, enabling dynamic and context-aware response generation. RAG is particularly effective for handling unstructured data and answering complex queries by leveraging relevant documents retrieved from a knowledge base [1].

* **Semantic Search with Dense Retrieval Models**: Dense retrieval frameworks, such as Dense Passage Retrieval (DPR), utilize embeddings to understand the semantic relationships between queries and documents, offering precise search results [38].

* **Question Answering Systems**: AI-powered systems leverage pre-trained large language models (LLMs) like GPT to answer user queries by extracting information directly from documents, providing concise and relevant responses [25].

* **Automated Metadata Extraction**: AI systems use natural language processing (NLP) to automatically extract metadata, such as titles, authors, and dates, from documents. This reduces manual tagging efforts and ensures consistency in document organization [42].

* **Document Clustering and Classification**: Machine learning models cluster or classify documents into categories based on content, streamlining organization and retrieval processes. These methods are often used in legal or medical document repositories for efficient navigation [46].

* **Optical Character Recognition (OCR) with NLP Integration**: OCR systems integrated with NLP extract text from scanned documents and analyze the content for further processing, making it accessible for search and retrieval in document management systems [47].

* **Chat-Based Interfaces**: AI-powered chatbots interact with users using natural language, offering personalized document search and assistance. These systems enhance accessibility and ease of use in corporate environments [48].

3.3 Limitations of Existing Works

While the reviewed works demonstrate significant advancements, several gaps remain:

* **Adaptability to Corporate Domains**: Most embedding and retrieval systems are evaluated on general-purpose datasets, making their adaptation to niche corporate domains challenging.

* **Integration of Retrieval and Generation**: Few studies explore the combined use of retrieval and generative models specifically for corporate environments.

* **Handling Unstructured Data**: Many approaches struggle with processing unstructured documents like PDFs with varied layouts, which are common in corporate settings.

3.4 Summary

This review highlights the evolution of document management systems, from traditional search strategies to AI-integrated solutions. While foundational works like keyword-based and Boolean search laid the groundwork for information retrieval, AI-driven methods such as semantic search and RAG have significantly advanced the field. By addressing the identified limitations, this thesis contributes a novel approach to enhancing corporate document management systems using embeddings, semantic search, and RAG frameworks.

Chapter 4

Methodology

This chapter outlines the methodology used in this study, covering data preparation, embedding models, LLMs, retrieval strategies, the RAG pipeline, and evaluation metrics for assessing retrieval accuracy and generation quality.

4.1 Introduction

The methodology for this project follows a systematic approach to implement a Retrieval-Augmented Generation (RAG) framework. Key steps include data preprocessing, selecting an optimal embedding model, evaluating retrieval strategies, generating responses with Large Language Models (LLMs), and assessing performance using evaluation metrics. These steps aim to address the challenges of managing and retrieving information from unstructured corporate documents, ensuring both precision and relevance.

Figure 4.1 illustrates the workflow, highlighting the additional contributions made on top of the standard RAG pipeline as part of this thesis.

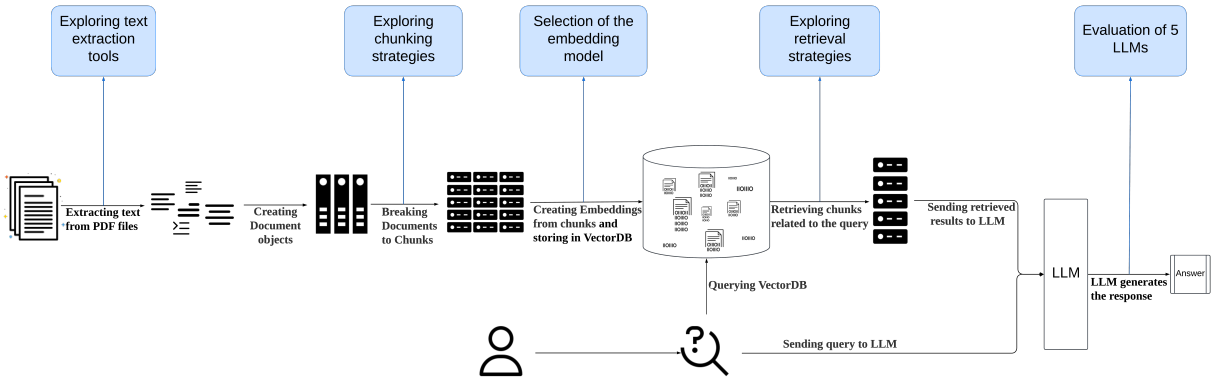


Figure 4.1: Blue sections represent the additional contributions made on top of the RAG pipeline in this thesis.

4.2 Dataset Collection and Organization

The dataset consisted of PDF documents provided by Flosslab, systematically organized in a hierarchical folder structure. The structure was designed to categorize documents based on projects and their specific document types:

1. **Project Level**:

   Each folder at the top level was named after a specific project, grouping all documents related to that project.

2. **Category Level**:

   Within each project folder, documents were further categorized into three distinct subfolders:

   * **Documentazione Tecnica di Progetto**: Technical project documentation, often containing detailed specifications, technical reports, and project plans.
   * **Offerta Commerciale**: Commercial offers, including financial proposals, pricing models, and related business details.
   * **Offerta Tecnica**: Technical offers, such as technical evaluations, requirements analyses, and feasibility studies.

3. **Document Level**:

   Each category subfolder contained the relevant PDF documents for that category and each document was uniquely formatted, reflecting the variability in style and complexity typical of corporate documentation.

4.3 Text Extraction and Structuring

The documents analyzed encompassed a diverse range of data types, including:

* **Text**: Narrative sections, headings, and technical descriptions, often varying in font styles, sizes, and alignment.
* **Tables**: Structured data ranging from simple grids to complex multi-page tables with merged cells and embedded annotations.
* **Pictures and Diagrams**: Visual content such as charts, schematics, and annotated illustrations, frequently used to complement textual data.
* **Code Snippets**: Segments of programming code formatted with indentation, syntax highlighting, or embedded within paragraphs.

The heterogeneity of these documents presented unique challenges during preprocessing, as no single extraction method could effectively handle all layouts. Tailored approaches were required for each data type to ensure accurate extraction while preserving structure and semantic integrity, making the preprocessing pipeline a critical aspect of the methodology.

4.3.1 Identifying Document Elements

Various methods were employed to extract and structure key elements from the documents, focusing on titles and subtitles, subscripts and superscripts, headers, footers, tables and the table of contents. These methods aimed to ensure accurate extraction while maintaining the structural integrity of the original content:

* **Titles:** Font size was used as a heuristic to identify titles, with larger font sizes (e.g., greater than 12 points) assumed to indicate headings or titles. Additional rules were implemented to differentiate titles from other elements, such as table of contents entries, by analyzing patterns within the first few pages of the document.
* **Subscripts and Superscripts:** Character positioning along the vertical axis (y-axis) relative to preceding characters was analyzed to identify subscripts and superscripts. This method aimed to detect variations in text alignment and distinguish these elements from normal text.
* **Tables:** Libraries such as `PyMuPDF` and `PDFPlumber` used to extract tabular data. These tools identified table structures by analyzing line separations, grid patterns, and text alignment. The goal was to preserve the logical structure of tables for downstream processing.
* **Headers and Footers:** Text located within the top and bottom 10% of each page was extracted to identify headers and footers. This included elements such as page numbers, document titles, and watermarks, which were isolated for potential removal or separate processing.
* **Table of Contents:** The presence of repetitive patterns and large-font entries on initial pages was analyzed to identify table of contents sections. These entries were flagged to differentiate them from the main content.

4.3.2 Text Extraction Approach

For extracting text from documents, several libraries were tested to evaluate their effectiveness in handling diverse layouts and formats. The tools tested included `PDFPlumber`, `PyMuPDF`, and `LangChain PyPDFLoader`. Each library was assessed for its ability to preserve structural features such as line breaks, text alignment, and overall document formatting.

Efforts were made to tag extracted elements during this process, such as marking titles, tables, and other key components with identifiable tags. These tags were intended to enhance downstream tasks like chunking and semantic analysis. However, the focus remained on creating a robust extraction pipeline that could handle a variety of document structures.

4.3.3 Document Structuring for Vector Database Storage

After extracting text, it was converted into *document objects* using tools provided by `LangChain`. Each document object comprised two components:

* **Metadata:** Included attributes such as project name, subject, and document name to enhance retrieval specificity.

* **Page Content:** Contained the cleaned and extracted text from the document.

4.3.4 Chunking for Retrieval Optimization

To improve retrieval precision and ensure the Large Language Model (LLM) generates responses from relevant and manageable content, the text was divided into smaller, structured units, referred to as *chunks*. Chunking was carefully designed to balance the size and context of each segment.

Challenges in Chunking

* If the *chunk size is too **small***, the context becomes fragmented, reducing the effectiveness of embeddings and retrieval. Additionally, semantic relationships between words may be disrupted or lost.

* If the *chunk size is too **large***, it risks exceeding the token limit of the embedding model, leading to incomplete or inefficient processing. Furthermore, the results may become diluted with extraneous information that is irrelevant to the query.

Chunking Methods

Three methods were tested for preparing chunks for embedding and retrieval, all utilizing the `LangChain Recursive Text Splitter`:

1. **Natural Breakpoints Method (Regex)**: A variation of the natural breakpoints method that treated separators as regular expressions.
   Using `is_separator_regex=True`, the separators were set to `["\n\n", "\n", " ", ""]`, enabling more flexible chunk splitting based on different text structures.

2. **Natural Breakpoints Method (Basic)**: Chunks were created using natural breakpoints, such as sentence boundaries, with the setup `separators=["\n"]`. This configuration split chunks based on newline characters to align with the logical structure of the text.

3. **Token-Based Sliding Window Method**: This approach employed a token-based sliding window with overlapping chunks to maintain context between segments. The setup included `is_separator_regex=False` to ensure that separators were retained while maintaining consistency across chunks.

Each method was evaluated for its ability to effectively structure text for embedding and retrieval.

At the end of the process, a unique `UUID` was assigned to each chunk and document. The metadata for each chunk included both a `chunkId` and a `documentId`, ensuring clear association between each chunk and its corresponding document.

4.4 Selection of the Embedding Model

To evaluate the performance of embedding models for this project, a structured methodology was designed to simulate real-world information retrieval tasks. This process included

generating queries, mapping them to document chunks, and ensuring traceability for reliable evaluation. Then measuring Mean Reciprocal Rank for the retrieved chunks for each query for each embedding model.

4.4.1 Embedding Evaluation Dataset: Query and Chunk Creation

The creation of the embedding evaluation dataset involved the following steps:

Chunk Selection

Two largest chunks from each document were selected to serve as representative segments for retrieval tasks. These chunks were chosen based on their size to maximize the coverage of meaningful content within each document.

Query Generation

Two queries were generated for each selected chunk to simulate typical information retrieval scenarios. The `mixtral:8x22b-instruct` model was employed to craft high-quality, contextually relevant prompts tailored to the content of each chunk.

UUID Assignment

To maintain traceability and facilitate efficient query management, each query was assigned a unique identifier (UUID). This ensured clear reference and organization throughout the evaluation process.

JSON Mapping of Relevant Documents

A structured JSON file was created to map each query to its corresponding chunks and associated documents. This mapping served as the ground truth for assessing embedding model performance, providing a clear and organized link between queries, chunks, and documents.

4.4.2 Storage of Documents in ChromaDB

The document data was stored in **ChromaDB**, a local vector database selected for its capability to securely store data on-premises, ensuring the confidentiality of company documents. The storage process in ChromaDB involved the following steps:

1. Ingesting the text from the documents.

2. Generating dense vector representations (embeddings) using the selected embedding model.

3. Storing these embeddings along with their associated metadata for efficient similarity-based retrieval.

4.4.3 Evaluation of Embedding Models

To identify the optimal embedding model for the project, multiple state-of-the-art models were evaluated:

* `mxbai-embed-large:latest`
* `nomic-embed-text:latest`
* `bge-m3-f16:latest`
* `multilingual-e5-large`

The evaluation methodology consisted of the following steps:

1. Generating embeddings for the documents using each embedding model.
2. Querying the vector database with the pre-generated queries.
3. Comparing the retrieved results against the ground truth defined in the JSON mapping file to assess accuracy and relevance.

The performance of each embedding model was evaluated using the **Mean Reciprocal Rank (MRR)** to assess retrieval effectiveness. Additionally, the efficiency of each model was measured using two key metrics: **embedding time**, which represents the time required to convert a document chunk into its dense vector representation, and **retrieval time**, the duration taken by the vector database to retrieve the relevant chunks. These metrics are critical for assessing the practical usability of embedding models, particularly in scenarios demanding real-time processing or large-scale document management.

4.5 RAG Implementation

After storing the document chunks in the vector store using the selected embedding model, the system retrieves relevant chunks by querying the vector database based on the input query. These retrieved chunks are then used as contextual information for the generation process. The RAG (Retrieval-Augmented Generation) pipeline was implemented as follows:

4.5.1 Retrieval

To determine the most effective retrieval strategy for this project, multiple approaches were tested and evaluated. The retrieval strategies explored were as follows:

* **Normal Retriever:** This method utilized similarity search to retrieve the top 10 chunks for each query based on embedding similarity.
* **Normal Reranked Retriever:** This approach extended the normal retriever by reranking the top 10 retrieved chunks using the `BAAI/bge-reranker-v2-m3` model. The reranker evaluated the relevance of each chunk more precisely to improve ranking accuracy.

* **LOTR (Merged Retriever):** A hybrid approach that combined similarity search with Maximal Marginal Relevance (MMR) to balance relevance and diversity in the retrieved results. Each retrieval method contributed 10 chunks, resulting in a total of 20 chunks retrieved. After merging the chunks, redundant chunks were removed to enhance the overall relevance of the result set. The remaining chunks were then reranked using the `BAAI/bge-reranker-v2-m3` model to ensure optimal relevance and contextual accuracy.

4.5.2 Generation

From the 10 retrieved and reranked chunks, the top 5 most relevant chunks were selected for response generation. These chunks were then used as contextual input for a selection of state-of-the-art Large Language Models (LLMs). The tested models included:

* `mixtral:8x7b`
* `Mistral-7B-v0.3`
* `llama3:8b-instruct-fp16`
* `llama3:70b-instruct-q4_0`
* `gemma2:9b`

4.5.3 Performance Evaluation of RAG

The performance of the Retrieval-Augmented Generation (RAG) pipeline was evaluated in retrieval and generation stages.

Retrieval Evaluation

Retrieval strategies were evaluated using the **Mean Reciprocal Rank (MRR)** metric, which measures the rank of the first relevant result for each query. This metric provides insights into the effectiveness of each retrieval strategy in retrieving the most relevant chunks for a given query.

Generation Evaluation

The generated responses for each LLM and retrieval strategy were evaluated using the **Continuous-Eval** framework, which employed the following metrics:

* **Deterministic Metrics:**
  - `Answer Correctness`
  - `Faithfulness`
* **LLM-Based Metrics:**
  - `LLM-Based Faithfulness`

- `LLM-Based Answer Correctness`
- `LLM-Based Style Consistency`

The `mixtral:8x22b-instruct` model was used to conduct the LLM-based evaluations, leveraging its advanced contextual understanding capabilities.

The average **generation time** for each query was recorded for all LLMs to evaluate their efficiency and the evaluation results were analyzed to determine the optimal combination of retrieval strategy and LLM, balancing metrics such as correctness, faithfulness, and style consistency.

Chapter 5

Results and Discussion

This chapter presents the results and discusses the insights gained from the experiments, providing a detailed analysis and evaluation of the findings.

5.1 Analysis of Text Extraction

The text extraction process was critical for preparing the dataset for embedding generation and retrieval tasks. Multiple tools were tested, including `PyPDF2`, `PDFPlumber`, and `PyMuPDF`, each offering unique strengths and limitations.

5.1.1 Evaluation of Extraction Tools

* `PyPDF2|:` Useful for basic text extraction but struggled with structural recognition, such as headings, tables, and formatting.

* `PDFPlumber|:` Excelled in extracting structured data, such as tables, but struggled with multi-column layouts, embedded images, and consistent detection of titles or subtitles.

* `PyMuPDF|:` The most effective tool for detecting sentence and paragraph boundaries, preserving document structure, and handling complex layouts like multi-column text and embedded elements.

5.1.2 Challenges in Text Extraction

Despite the tools' capabilities, several challenges were observed during the extraction process:

* **Breakpoints:** `PyMuPDF` was the only tool to reliably detect sentence and paragraph boundaries. Other tools often failed to maintain logical text flow.

* **Headers and Footers:** A method focusing on the top and bottom 10% of each page, implemented using the `page.cropbox` feature from `PyMuPDF`, effectively identified headers and footers. This approach successfully removed extraneous content, such as page numbers and watermarks, while preserving the main body of the text.

* **Tables:** While `PDFPlumber` was effective for simple table layouts, it struggled with complex tables containing merged cells, nested structures, or spanning multiple pages. Misalignment and partial extractions were common.

* **Headings and Titles:** Titles were detected based on font size, which was unreliable for documents with inconsistent formatting or large font sizes in non-title elements, such as tables of contents.

* **Subscripts and Superscripts:** Identifying subscripts and superscripts using vertical position (y-axis) heuristics proved unreliable due to encoding inconsistencies and variations in document styles.

5.1.3 Overall Comparison and Selection

The quality of text extraction was manually reviewed across 134 corporate documents, with less than **50%** of the documents successfully processed using the implemented methods. Key observations included:

* **Breakpoints:** `PyMuPDF` reliably detected sentence and paragraph boundaries, while other tools often produced fragmented text.

* **Tables:** Many tables, particularly those with complex layouts, were partially extracted, misaligned, or entirely missed.

* **Subscripts and Superscripts:** Detection of subscripts and superscripts was inconsistent due to heuristic limitations, although manual corrections were relatively straightforward.

* **Titles and Formatting:** Titles and subtitles were inconsistently identified, with frequent misclassification caused by reliance on font size alone.

These results highlighted significant limitations in the current extraction techniques and underscored the need for more advanced tools tailored to structured data extraction from diverse corporate document formats.

Given the heterogeneity of document layouts, tailoring the extraction process for each element of a document was found to be neither efficient nor effective. Instead, a generalized framework based on `PyMuPDF` was selected for its robust layout-aware capabilities and its ability to retain meaningful structure across a variety of document types. The extracted text was then manually processed to remove special characters and cleaned to eliminate unnecessary breakpoints, such as those caused by sentence wrapping, ensuring a more coherent and structured output.

5.2 Embedding Model Evaluation

The embedding models were evaluated using three key metrics: **Mean Reciprocal Rank (MRR)** to assess ranking effectiveness, **Average Embedding Time**, and **Average Retrieval Time** to measure efficiency.

5.2.1 Mean Reciprocal Rank (MRR)

The MRR metric evaluates the effectiveness of embedding models in ranking relevant documents by considering the position of the first relevant result for each query.

Figure 5.1 illustrates the MRR scores for the evaluated embedding models:

* *bge-m3-f16* achieved the highest MRR score of **0.56**, demonstrating good retrieval accuracy.
* *multilingual-e5-large* followed with an MRR score of **0.24**, reflecting moderate performance.
* *nomic-embed-text* and *mxbai-embed-large* scored **0.10** and **0.05**, respectively, indicating limited effectiveness for this application.

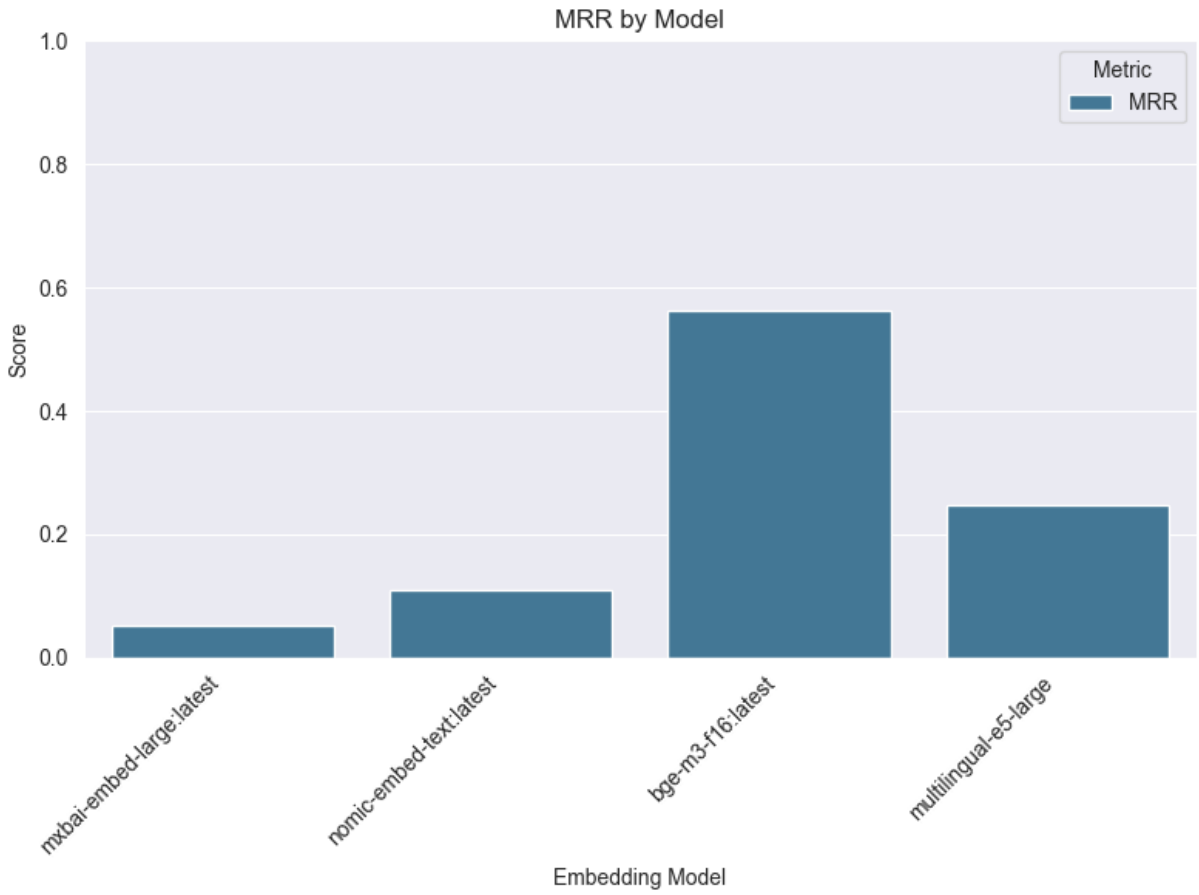


Figure 5.1: Mean Reciprocal Rank (MRR) Results for Evaluated Embedding Models

5.2.2 Efficiency Analysis

Embedding Time Results

The embedding duration reflects the time required to generate embeddings for all the documents using each model. Figure 5.2 presents the embedding durations for the evaluated models:

* *bge-m3-f16* demonstrated the shortest embedding time of **212.19 seconds**, highlighting its computational efficiency.
* *multilingual-e5-large* followed closely with **239.96 seconds**.
* *nomic-embed-text* and *mxbai-embed-large* exhibited significantly higher embedding durations of **2600.69 seconds** and **3131.02 seconds**, respectively.

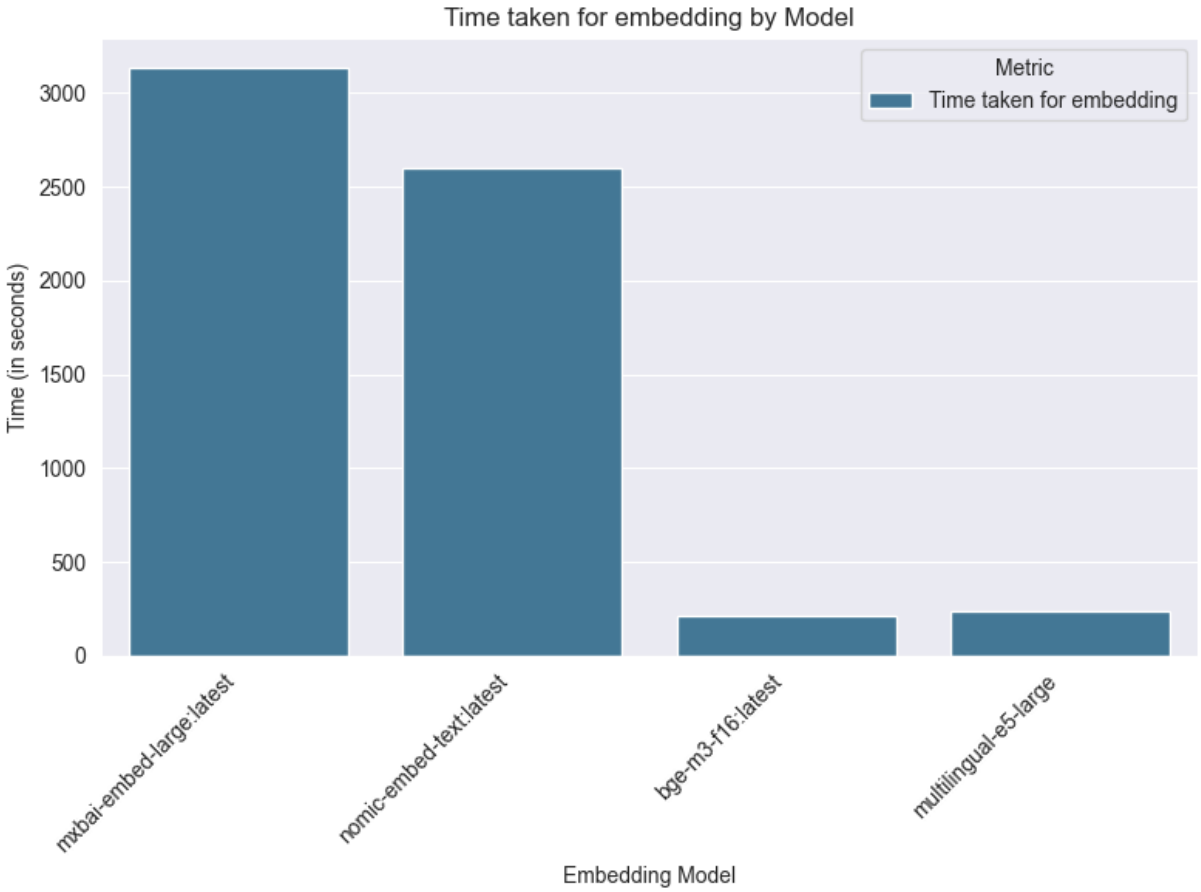


Figure 5.2: Embedding Time for Evaluated Models

Retrieval Time Results

Retrieval time measures the average duration required to retrieve relevant chunks for a query, reflecting the practical efficiency of each embedding model. Table 5.1 and Figure 5.3 present the retrieval times for the evaluated models:

* ***bge-m3-f16*:** The shortest retrieval time (**1.97 seconds**), combined with the highest MRR score, highlights its balance of accuracy and efficiency.
* *multilingual-e5-large* performed moderately well with a retrieval time of **2.03 seconds**.
* *nomic-embed-text* and *mxbai-embed-large* showed significantly higher retrieval times of **4.35 seconds** and **6.26 seconds**, respectively, making them less suitable for real-time applications.

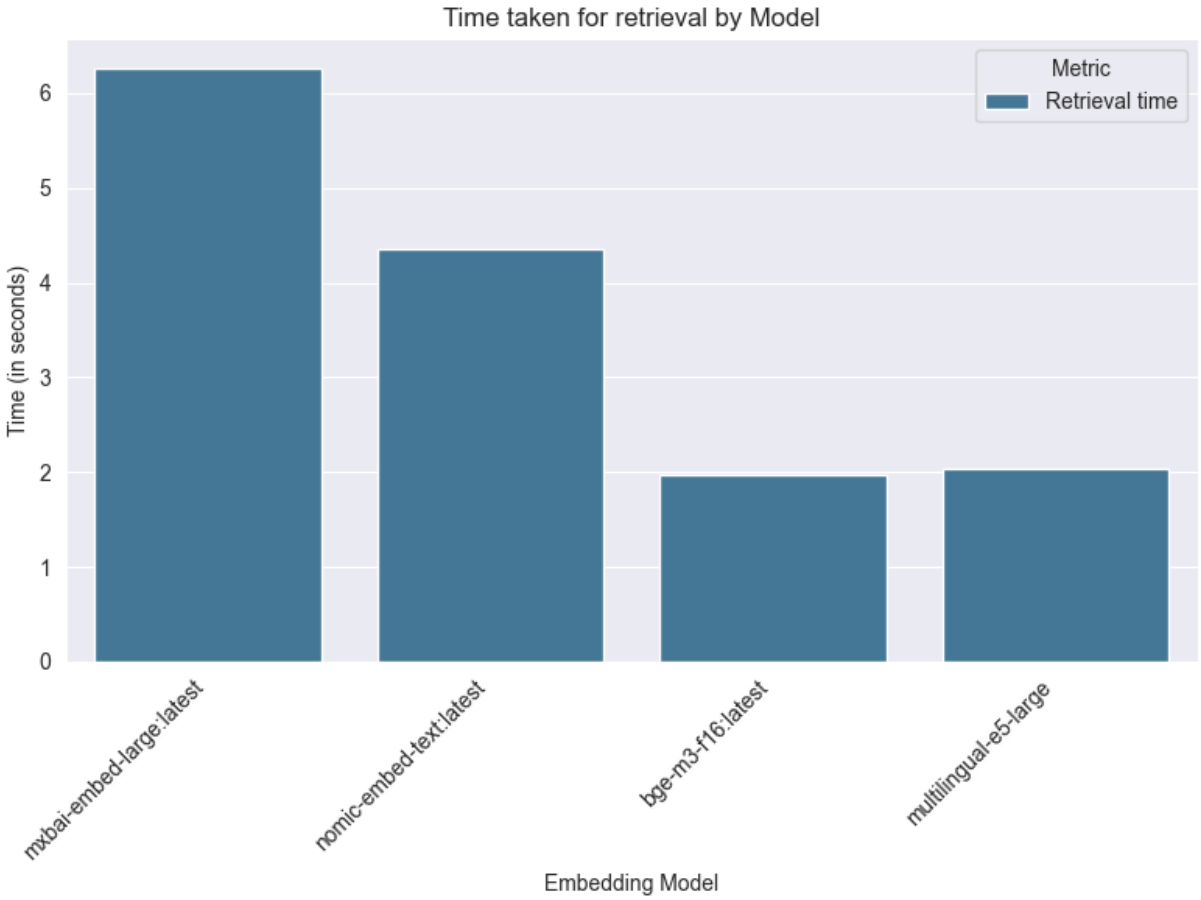


Figure 5.3: Average Retrieval Time for Evaluated Embedding Models

5.2.3 Summary of Results

Table 5.1 consolidates the evaluation metrics, highlighting the superior performance of *bge-m3-f16* across all key metrics.

Table 5.1: Evaluation metrics for embedding models, including embedding duration (in seconds), MRR score, and average retrieval time (in seconds). A higher MRR score indicates better retrieval performance, while lower embedding and retrieval times signify greater efficiency. The best results for each metric are highlighted in **bold**

Embedding Model	Embedding Duration (s)	MRR Score	Average Retrieval Time (s)
`mxbai-embed-large`	3131.02	0.05	6.26
`nomic-embed-text`	2600.69	0.10	4.35
`bge-m3-f16`	**212.19**	**0.56**	**1.97**
`multilingual-e5-large`	239.96	0.24	2.03

5.2.4 Conclusion

The *bge-m3-f16* model stands out due to its exceptional performance, attributed to the following key strengths:

* **Semantic Precision:** Accurately captures fine-grained semantic relationships between queries and documents.
* **Domain Robustness:** Effectively adapts to diverse corporate document types, ensuring consistent performance.
* **Efficient Representation:** Strikes an ideal balance between computational efficiency and retrieval accuracy.

With its high retrieval accuracy (**MRR = 0.56**) and remarkable computational efficiency (**Embedding Time = 212.19 seconds**, **Retrieval Time = 1.97 seconds**), the *bge-m3-f16* model emerges as the optimal choice for this application. Its performance makes it the most suitable embedding model for the project, effectively meeting the demands of accurate and efficient retrieval.

5.3 Analysis of Retrieval Strategies

The retrieval strategies were evaluated using the **Mean Reciprocal Rank (MRR)** metric to measure their ranking quality. The results highlight the differences in effectiveness among the tested strategies.

5.3.1 MRR Results for Retrieval Strategies

Figure 5.4 visually compares the MRR scores for the evaluated retrieval strategies. The corresponding numerical results are summarized in Table 5.2.

Table 5.2: MRR Results for Retrieval Strategies

Retrieval Strategy	**MRR Score**
Similarity Search	0.59
Similarity Search + Semantic Reranking	0.69
Merged Retriever (Similarity Search + MMR + Semantic Reranking)	0.74

The results demonstrate the significant impact of retrieval strategies on the system's performance:

1. Similarity Search (Baseline)

Similarity Search achieved an MRR score of **0.59**, establishing the baseline for retrieval performance. It is highly efficient and computationally lightweight, making it suitable for straightforward retrieval tasks. However, it lacks the sophistication required to handle complex or nuanced queries effectively.

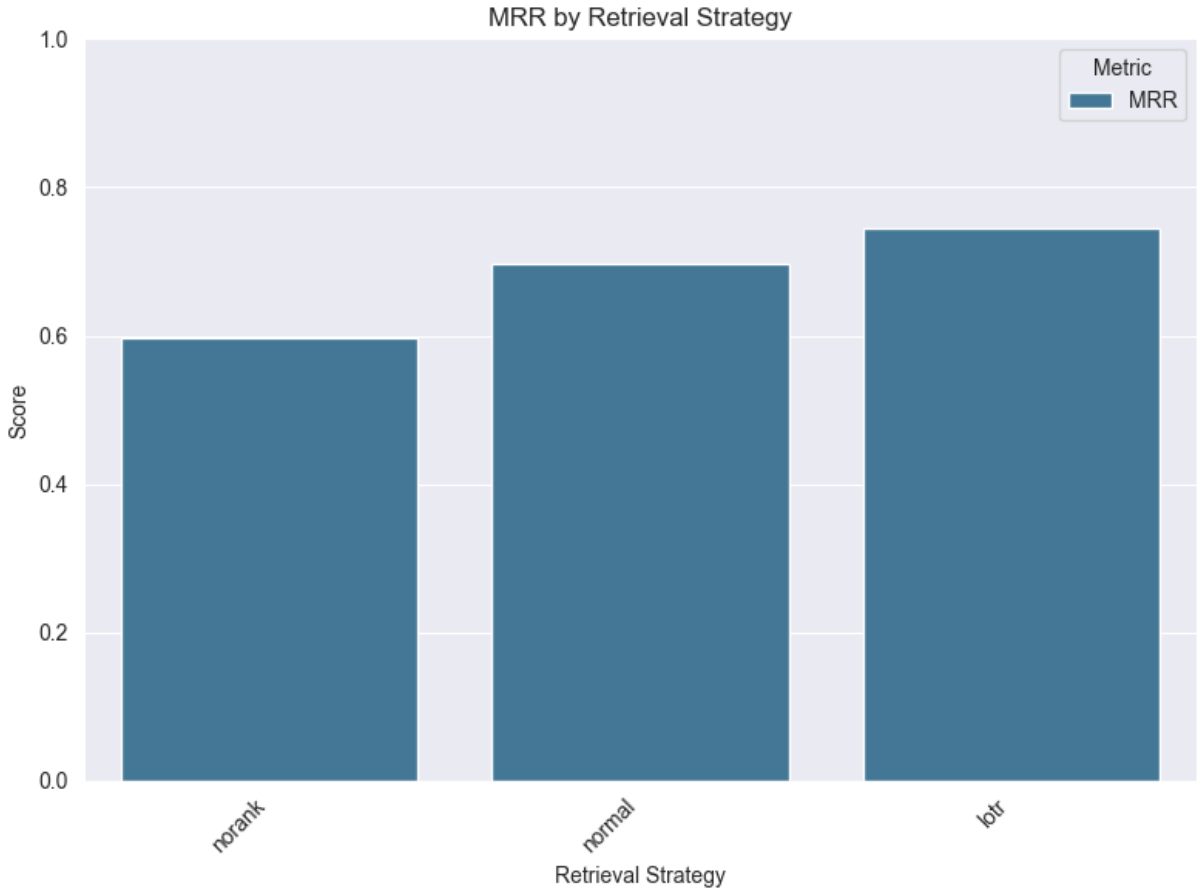


Figure 5.4: Comparison of MRR Scores for Retrieval Strategies. The first bar, labeled `norank`, represents similarity search. The middle bar, labeled `normal`, corresponds to similarity search with semantic reranking. The right bar `lotr` represents the merged retriever approach, combining similarity search and semantic reranking.

2. Semantic Reranking

Semantic Reranking improved the MRR score to **0.69** by incorporating a reranking step after retrieval. This approach significantly enhances precision, particularly for targeted or specific queries requiring fine-grained relevance. However, the added computational cost may pose challenges in resource-constrained environments.

3. Maximal Marginal Relevance (MMR)

The approach with MMR delivered the highest MRR score of **0.74**, outperforming other strategies. By balancing relevance and diversity, it excels in handling ambiguous or exploratory queries. While slightly more computationally intensive, the improved quality of retrieval justifies the trade-off.

5.3.2 Insights and Conclusions

The findings underscore the importance of tailoring retrieval strategies to the application's needs:

* **Baseline Similarity Search:** While fast and efficient, it is best suited for scenarios with simple retrieval requirements and limited computational resources.

* **Semantic Reranking:** Ideal for applications where precision is critical and computational resources are sufficient to accommodate additional processing.

* **MMR with Semantic Reranking:** Offers the best balance of relevance and diversity, making it the preferred strategy for applications requiring comprehensive and exploratory retrieval. However, this approach may not be suitable for real-time applications due to its increased computational demands and complexity.

These findings validate the critical role of retrieval strategy selection in enhancing response quality for RAG systems, ensuring both precision and adaptability in information retrieval while balancing computational efficiency.

5.4 LLM Performance in RAG

This section evaluates the performance of the tested Large Language Models (LLMs) within the Retrieval-Augmented Generation (RAG) system. The evaluation combines `deterministic` and `LLM-based` metrics to assess the quality of the generated responses.

5.4.1 Deterministic Metrics

Table 5.3 presents the average correctness and faithfulness scores for the tested LLMs. These averages are computed by aggregating all submetrics within each metric category for each LLM and retrieval strategy. This provides an overall evaluation of model performance. In subsequent sections, each submetric will be analyzed individually to provide a more detailed understanding of the models' strengths and weaknesses.

Table 5.3: Average Correctness and Faithfulness for LLMs

Model	Average Correctness	Average Faithfulness
`mixtral:8x7b`	0.000523	0.640781
`Mistral-7B-v0.3`	0.000351	0.794784
`llama3:8b-instruct-fp16`	0.000291	0.758925
`llama3:70b-instruct-q4_0`	0.000611	0.798557
`gemma2:9b`	0.000518	0.608781

The `llama3:70b-instruct-q4_0` model demonstrated superior performance, achieving the highest scores for correctness (0.000611) and faithfulness (0.798557). Its large-scale parameterization enables nuanced and accurate responses to complex queries, while the Q4_0 quantization technique balances computational efficiency and high accuracy, making it scalable for real-world applications.

However, deterministic metrics such as **ROUGE**, **Token Overlap**, and **BLEU** pose challenges in evaluating correctness and faithfulness, as they rely on exact string matching. This approach penalizes models for variations in phrasing, synonyms, or sentence restructuring, even when the responses are semantically correct. Additionally, concise responses

that omit less critical details, though accurate, often result in lower recall and overlap scores. While these metrics provide some insights, their reliance on strict token matching limits their ability to fully capture semantic correctness. Using LLM-based metrics may offer a more robust evaluation, as will be explored further.

Deterministic Answer Correctness

The deterministic correctness metrics evaluate the ability of Large Language Models (LLMs) to generate answers that align closely with the ground truth. These metrics include ROUGE-L (Recall, Precision, and F1), Token Overlap (Recall and Precision), and BLEU Score, which collectively measure exact token matching between the generated answer and the reference answers.

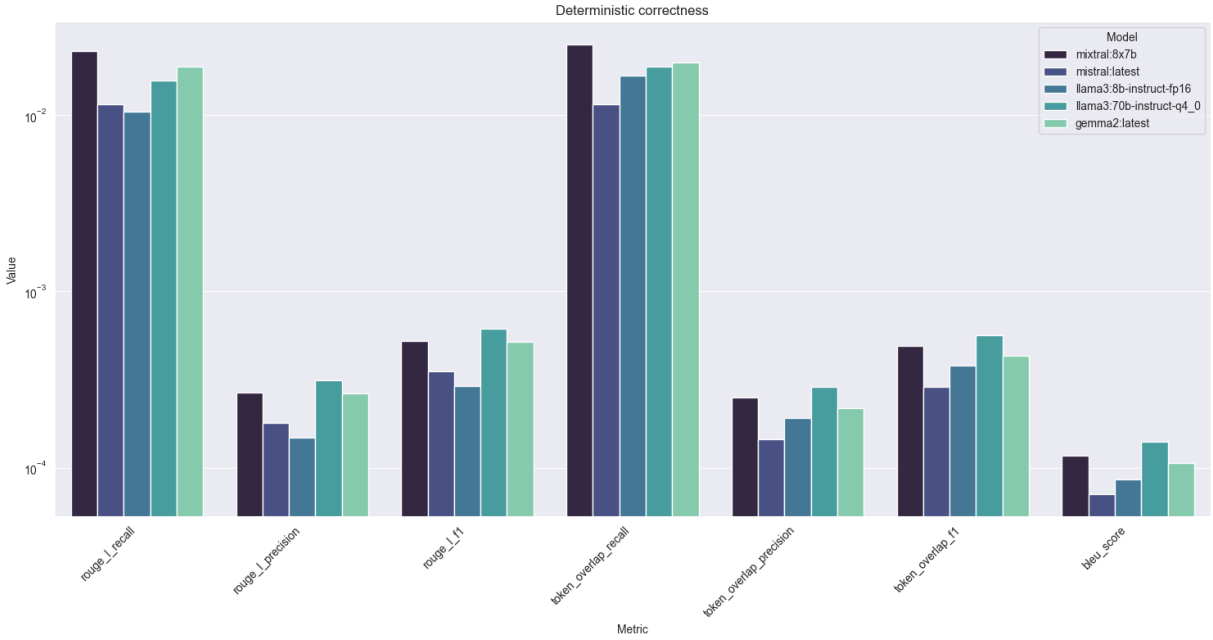


Figure 5.5: Deterministic Answer Correctness metrics grouped by LLMs. The figure illustrates the performance of each model across metrics such as ROUGE-L Recall, Token Overlap Recall, and BLEU Score.

Table 5.4 summarizes the average deterministic correctness metrics for the tested LLMs, and Figure 5.5 provides a visual representation of these metrics. These metrics highlight the following insights:

* **ROUGE-L Recall:** Measures the longest common subsequence (LCS) between the generated answer and the ground truth. Among the models, `mixtral:8x7b` achieved the highest recall (0.0232), indicating better alignment with the ground truth.
* **Token Overlap Recall:** Evaluates the proportion of tokens in the generated answer that match the ground truth. `mixtral:8x7b` again led with a score of 0.0253.
* **BLEU Score:** Assesses n-gram precision, considering brevity penalties for shorter answers. All models performed similarly on this metric, with scores around 0.0001, indicating challenges in exact n-gram matching.

Table 5.4: Average Deterministic Correctness Metrics for Each Model. Metrics include ROUGE-L (Recall, Precision, and F1), Token Overlap (Recall and Precision), and BLEU Score.

Model	ROUGE-L Recall	ROUGE-L Precision	ROUGE-L F1	Token Overlap Recall	Token Overlap Precision	BLEU Score
mixtral:8x7b	0.0232	0.0003	0.0005	0.0253	0.0002	0.0001
Mistral-7B-v0.3	0.0116	0.0002	0.0004	0.0116	0.0001	0.0001
llama3:8b-instruct-fp16	0.0105	0.0001	0.0003	0.0168	0.0002	0.0001
llama3:70b-instruct-q4_0	0.0158	0.0003	0.0006	0.0190	0.0003	0.0001
gemma2:9b	0.0190	0.0003	0.0005	0.0200	0.0002	0.0001

Discussion and Insights: The results reveal that deterministic correctness metrics, such as ROUGE-L Recall and Token Overlap Recall, effectively highlight the performance differences between models of varying parameterization. Larger models, like `llama3:70b-instruct-q4_0`, consistently outperform smaller ones, such as `gemma2:9b` and `Mistral-7B-v0.3`, due to their superior ability to capture complex contextual nuances. However, BLEU scores remain uniformly low across all models, reflecting its limitation as a metric for assessing semantically accurate but structurally varied answers.

Smaller models struggle to align with ground truth responses, as evidenced by their lower scores, which can be attributed to their reduced capacity for processing intricate contexts. The concise nature of responses generated by LLMs further affects recall scores, particularly for smaller models. These findings emphasize the need for additional qualitative or context-aware metrics to better assess the nuances of generated content in open-ended tasks.

Deterministic Faithfulness

Deterministic faithfulness metrics evaluate how well the generated responses adhere to the retrieved content. These metrics include ROUGE Faithfulness, Token Overlap Faithfulness, and BLEU Faithfulness, as well as sentence-level precision scores. Figure 5.6 provides a comparative visualization of the models' performance, and Table 5.5 summarizes the results.

Discussion and Analysis: Models with larger parameterization, such as `llama3:70b-instruct-q4_0` and `Mistral-7B-v0.3`, consistently achieve higher ROUGE and Token Overlap Faithfulness scores, showcasing their superior ability to generate responses closely aligned with retrieved content. These models effectively utilize their advanced contextual understanding to ground responses in the retrieved chunks. Although BLEU Faithfulness scores remain low across all models, larger models, including `llama3:8b-instruct-fp16` and `llama3:70b-instruct-q4_0`, demonstrate better phrase-level n-gram alignment. However, the limitations of BLEU in evaluating semantically accurate but structurally varied responses underscore its inadequacy for assessing nuanced LLM outputs.

Sentence-level metrics further highlight the precision of larger models in maintaining

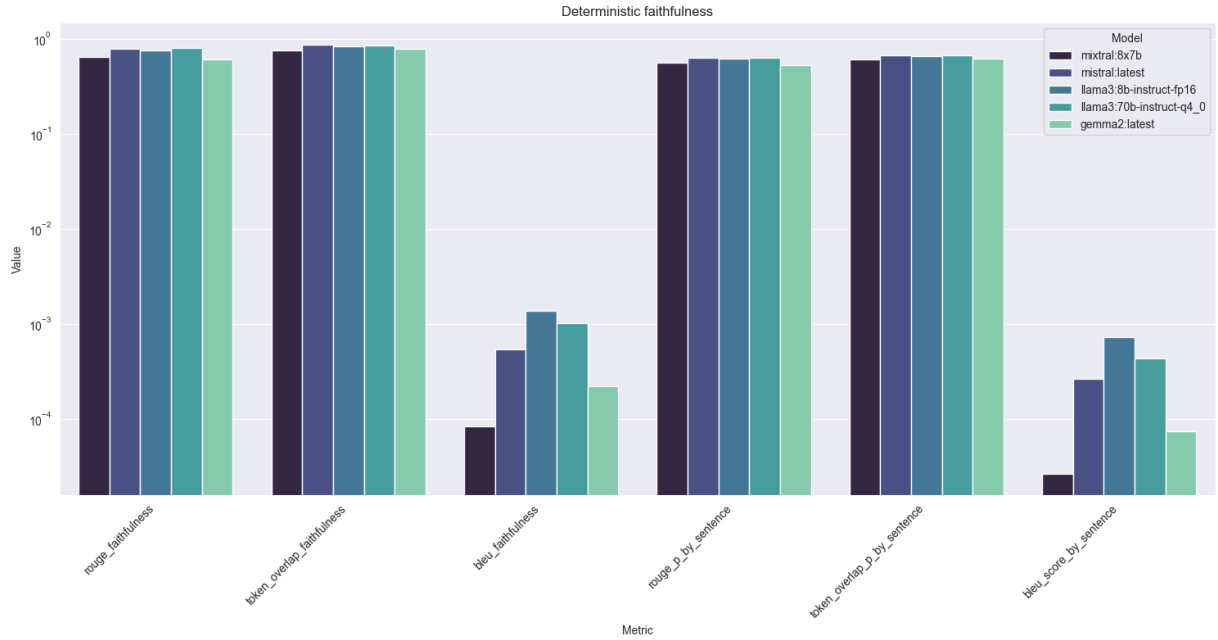


Figure 5.6: Deterministic Faithfulness Metrics Grouped by LLMs. This figure highlights the adherence of responses to the retrieved content, based on metrics such as ROUGE Faithfulness and Token Overlap Faithfulness.

Table 5.5: Faithfulness Metrics for Each Model. This table includes ROUGE Faithfulness, Token Overlap Faithfulness, BLEU Faithfulness, and their respective sentence-level precisions.

Model	ROUGE Faithfulness	Token Overlap Faithfulness	BLEU Faithfulness	ROUGE Precision by Sentence	Token Overlap Precision by Sentence	BLEU Score by Sentence
mixtral:8x7b	0.6408	0.7628	0.000083	0.5632	0.6097	0.000027
Mistral-7B-v0.3	0.7948	0.8794	0.000541	0.6389	0.6721	0.000266
llama3:8b-instruct-fp16	0.7589	0.8394	0.001369	0.6232	0.6611	0.000724
llama3:70b-instruct-q4_0	0.7986	0.8611	0.001025	0.6285	0.6692	0.000433
gemma2:9b	0.6088	0.7928	0.000220	0.5280	0.6206	0.000074

adherence to specific retrieved sentences. Models such as `mixtral:8x22b-instruct` and `llama3:70b-instruct-q4_0` efficiently utilize retrieved chunks, achieving high grounding accuracy. In contrast, smaller models like `gemma2:9b` and `mixtral:8x7b` struggle with contextual reasoning and tend to rely on broader generalizations, resulting in lower faithfulness scores. These findings validate the reliability of larger models for tasks requiring precise grounding, while smaller models reflect the trade-offs between computational efficiency and contextual fidelity.

5.4.2 LLM-Based Metrics

This section evaluates the performance of the tested LLMs based on correctness, faithfulness, and style consistency, as measured by LLM-based metrics. Table 5.6 presents the average scores for each metric across all tested models. These values represent the averages obtained from the three retrieval strategies for each metric and LLM, providing a comprehensive view of model performance under varying retrieval conditions. The analysis that follows examines the performance across all metrics, highlighting key strengths and limitations of the models.

Table 5.6: LLM-Based Metric Averages for Each Model

Model	Correctness	Faithfulness	Style Consistency
`mixtral:8x7b`	0.8201	0.8432	0.7659
`Mistral-7B-v0.3`	0.7870	0.8452	0.7382
`llama3:8b-instruct-fp16`	0.8066	0.8705	0.7566
`llama3:70b-instruct-q4_0`	0.8084	0.8990	0.7792
`gemma2:9b`	0.8383	0.8674	0.8239

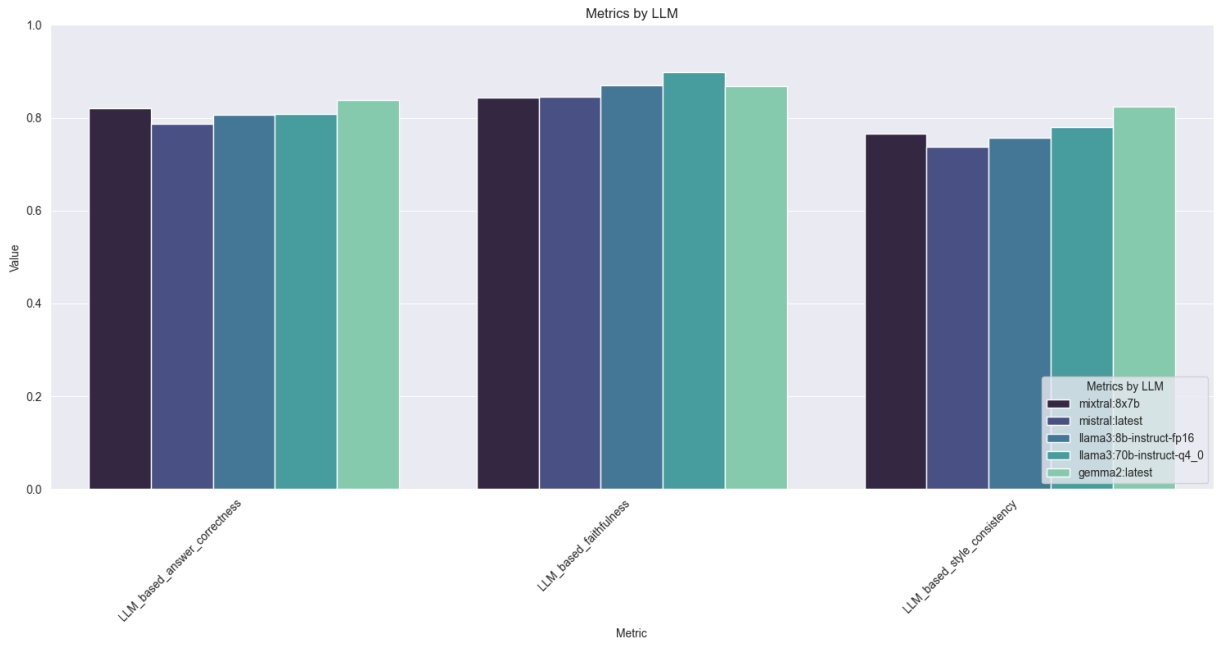


Figure 5.7: Comprehensive LLM-Based Metrics Across Models.

Correctness scores assess the models' ability to generate factually accurate responses. Among the tested models, `gemma2:9b`, despite being a smaller model, achieved the highest correctness score (0.8383), closely followed by `mixtral:8x7b` (0.8201). Larger models, such as `Mistral-7B-v0.3`, scored slightly lower, indicating challenges in maintaining factual consistency, particularly when handling more complex queries.

Faithfulness evaluates the grounding of responses in the retrieved content.

The `llama3:70b-instruct-q4_0` model achieved the highest score (0.8990), demonstrating exceptional alignment of its responses with the retrieved context.

Smaller models, including `gemma2:9b` and `llama3:8b-instruct-fp16`, also performed well, showcasing their ability to generate contextually grounded responses that closely adhere to the source material.

Style Consistency measures the coherence and alignment of responses with a desired tone. Although a smaller model, `gemma2:9b` excelled in this metric (0.8239), highlighting its ability to maintain a formal and consistent tone throughout its responses. In contrast, other smaller models, such as `Mistral-7B-v0.3`, struggled with stylistic alignment, likely due to reduced parameterization and limited adaptability to specific styles.

LLM-Based Correctness

The *lotr* retrieval strategy consistently outperformed *norank* and *normal* strategies across all tested models, leading to higher correctness scores by enhancing the relevance of retrieved chunks. Among the models, `gemma2:9b` and `llama3:70b-instruct-q4_0` achieved the highest correctness scores, showcasing their advanced contextual reasoning and ability to accurately respond to complex queries.

In contrast, smaller models like `Mistral-7B-v0.3` struggled with reasoning and factual correctness, particularly when using the less optimized *norank* strategy. This highlights their limited capability to leverage retrieved content effectively for intricate queries.

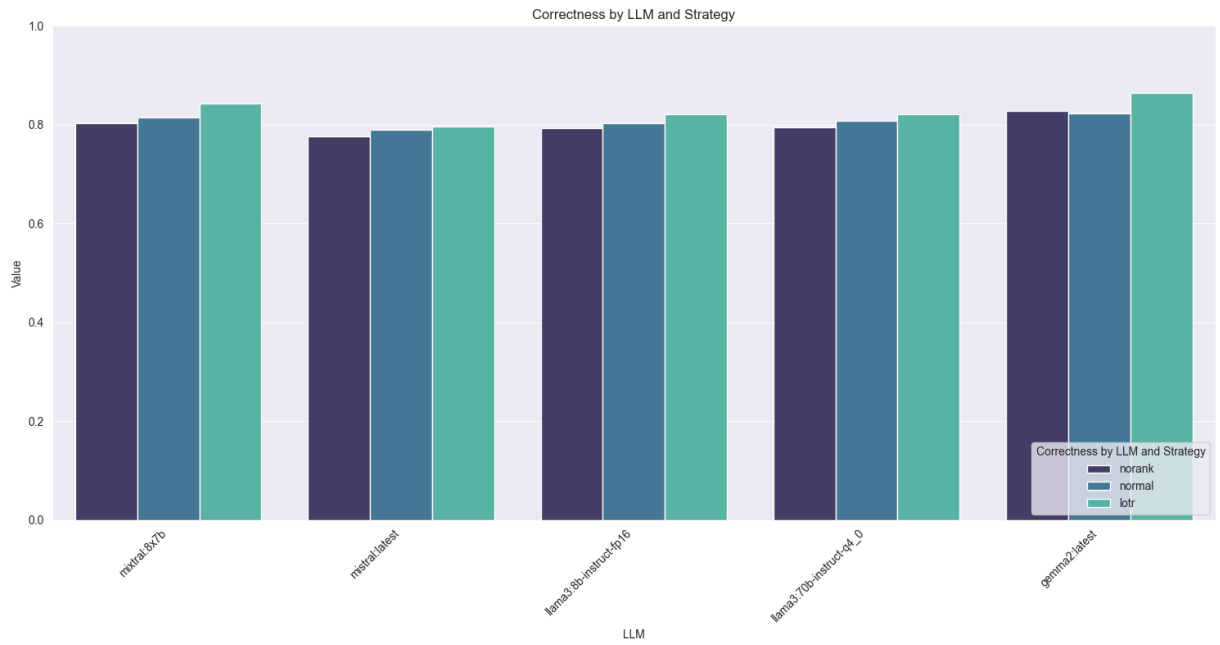


Figure 5.8: LLM-Based Correctness by Model and Strategy. The chart illustrates the impact of retrieval strategies on model correctness.

LLM-Based Faithfulness

The *lotr* strategy consistently improved faithfulness scores across all tested models by enhancing the quality of retrieved chunks. Larger models, such as `llama3:70b-instruct-q4_0` and `gemma2:9b`, achieved the highest faithfulness scores, indicating their ability to maintain strong alignment with retrieved content and minimize hallucinations.

Table 5.7: Correctness Metrics by Retrieval Strategy

Model	Norank	Normal	LOTR
`mixtral:8x7b`	0.8038	0.8139	0.8427
`Mistral-7B-v0.3`	0.7757	0.7890	0.7961
`llama3:8b-instruct-fp16`	0.7939	0.8036	0.8222
`llama3:70b-instruct-q4_0`	0.7951	0.8088	0.8214
`gemma2:9b`	0.8279	0.8226	0.8644

Smaller models, including `Mistral-7B-v0.3`, displayed lower faithfulness scores, particularly with less optimized retrieval strategies. This reflects their limitations in grounding responses effectively in the retrieved information.

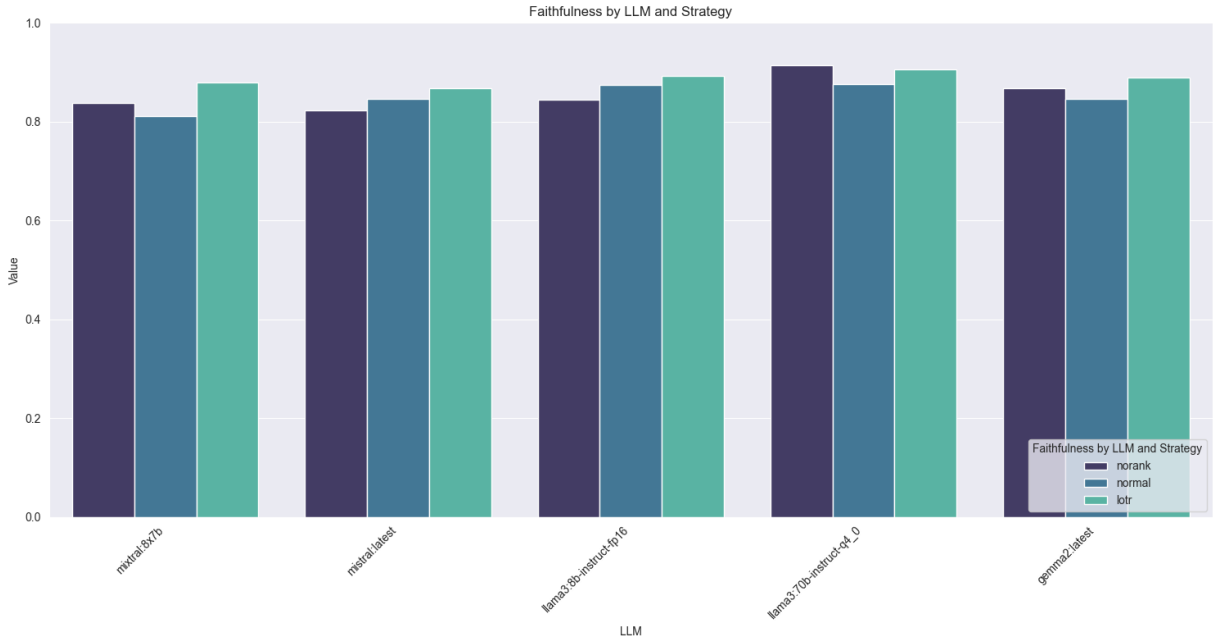


Figure 5.9: LLM-Based Faithfulness by Model and Strategy.

Table 5.8: Faithfulness Metrics by Retrieval Strategy

Model	Norank	Normal	LOTR
`mixtral:8x7b`	0.8386	0.8107	0.8801
`Mistral-7B-v0.3`	0.8228	0.8454	0.8675
`llama3:8b-instruct-fp16`	0.8449	0.8738	0.8927
`llama3:70b-instruct-q4_0`	0.9146	0.8770	0.9054
`gemma2:9b`	0.8671	0.8454	0.8896

LLM-Based Style Consistency

The *lotr* strategy significantly enhanced stylistic alignment for all models by providing contextually rich chunks, enabling them to produce coherent and well-structured responses.

`gemma2:9b` achieved the highest style consistency scores across all strategies, particularly excelling with the *lotr* strategy (0.8333), reflecting its ability to align responses with corporate tone and maintain coherence.

Smaller models, such as `Mistral-7B-v0.3`, exhibited lower style consistency, often struggling with sentence transitions and formatting when handling complex queries. This highlights the limitations of smaller models in maintaining formal writing styles and producing cohesive responses.

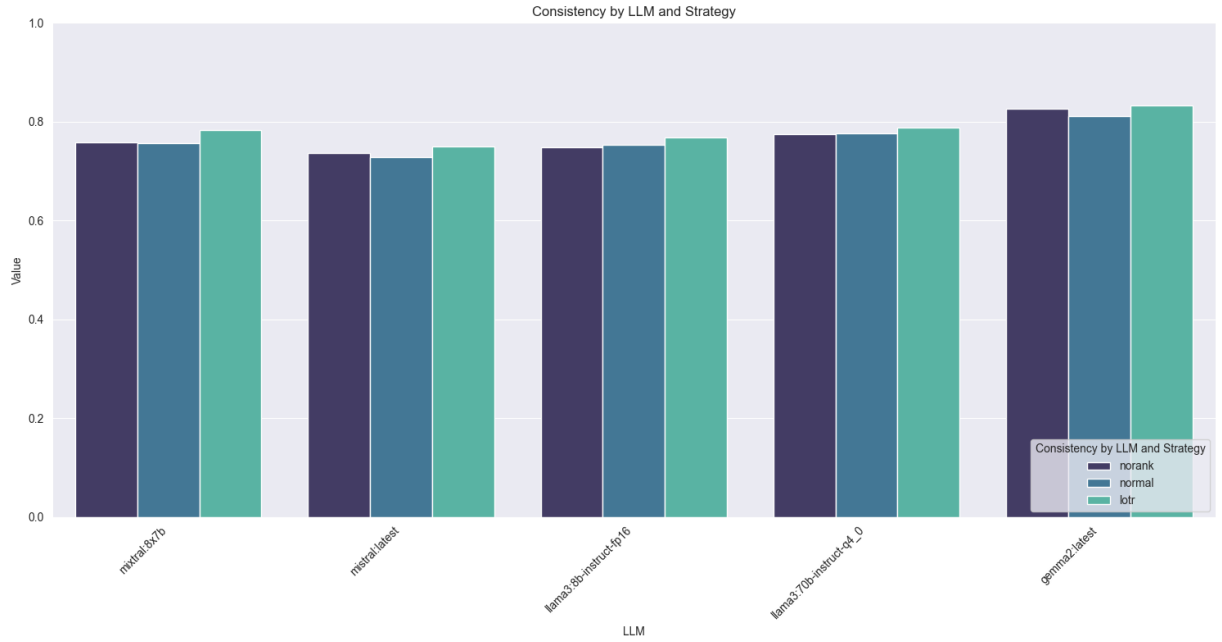


Figure 5.10: LLM-Based Style Consistency by Model and Strategy.

Table 5.9: Style Consistency Metrics by Retrieval Strategy

Model	Norank	Normal	LOTR
`mixtral:8x7b`	0.7591	0.7560	0.7826
`Mistral-7B-v0.3`	0.7363	0.7282	0.7503
`llama3:8b-instruct-fp16`	0.7482	0.7529	0.7687
`llama3:70b-instruct-q4_0`	0.7741	0.7760	0.7876
`gemma2:9b`	0.8270	0.8113	0.8333

Summary of Findings: Across all evaluation metrics, the *lotr* retrieval strategy consistently enhanced performance by improving the quality of retrieved chunks. Larger models like `llama3:70b-instruct-q4_0` and `gemma2:9b` excelled in correctness, faithfulness, and style consistency, making them the most reliable for complex, nuanced tasks. In contrast, smaller models, such as `Mistral-7B-v0.3`, highlighted the trade-off between computational efficiency and output quality, struggling to maintain high accuracy and alignment in more demanding scenarios.

5.4.3 Average Response Time of LLMs

The table (Table 5.10) and figure (Figure 5.11) present the average response times of various large language models (LLMs), measured in seconds. Lightweight models, such as `mistral_llm` and `gemma2_llm`, demonstrate faster response times compared to larger models like `mixtral_8x22b` and `llama70b_llm`, which exceed 11 seconds on average due to their higher complexity and parameter counts.

These findings highlight the trade-offs between model complexity and response efficiency. While larger models provide advanced capabilities, they require more computational resources and time, making lightweight models a better choice for applications prioritizing speed and resource efficiency. This analysis aids in selecting the most suitable LLM based on specific application needs.

Table 5.10: Average Response Time for LLMs

Model	Average Response Time (s)
`mixtral:8x7b`	5.6853
`Mistral-7B-v0.3`	2.6932
`llama3:8b-instruct-fp16`	4.7071
`llama3:70b-instruct-q4_0`	11.5303
`gemma2:9b`	2.8846

5.5 Overview

This chapter evaluated the Retrieval-Augmented Generation (RAG) system, focusing on its key components: text extraction, embedding models, retrieval strategies, and response generation. Performance was assessed using deterministic and LLM-based metrics, highlighting several key findings:

* **Text Extraction:** Extracting structured and coherent content from diverse corporate documents remains a challenge. Preprocessing helped clean outputs, but issues like inconsistent title and table detection underscore the need for adaptive extraction methods tailored to varying document structures.

* **Embedding Models:** The choice of embedding model greatly impacts retrieval quality. `bge-m3-f16` emerged as the best-performing model with the highest Mean Reciprocal Rank (MRR) and shortest embedding and retrieval times, effectively balancing semantic precision and computational efficiency.

* **Retrieval Strategies:** The Similarity Search with Maximal Marginal Relevance (MMR), followed by semantic reranking, consistently outperformed other strategies by providing a balanced mix of relevance and diversity in retrieved content. However, its computational demands make it less suitable for real-time applications, necessitating optimization for specific use cases.

* **Model Performance:** Larger models like `llama3:70b-instruct-q4_0` demonstrated superior correctness and faithfulness due to their advanced contextual reasoning. Smaller models, such as `gemma2:9b`, excelled in style consistency, making them

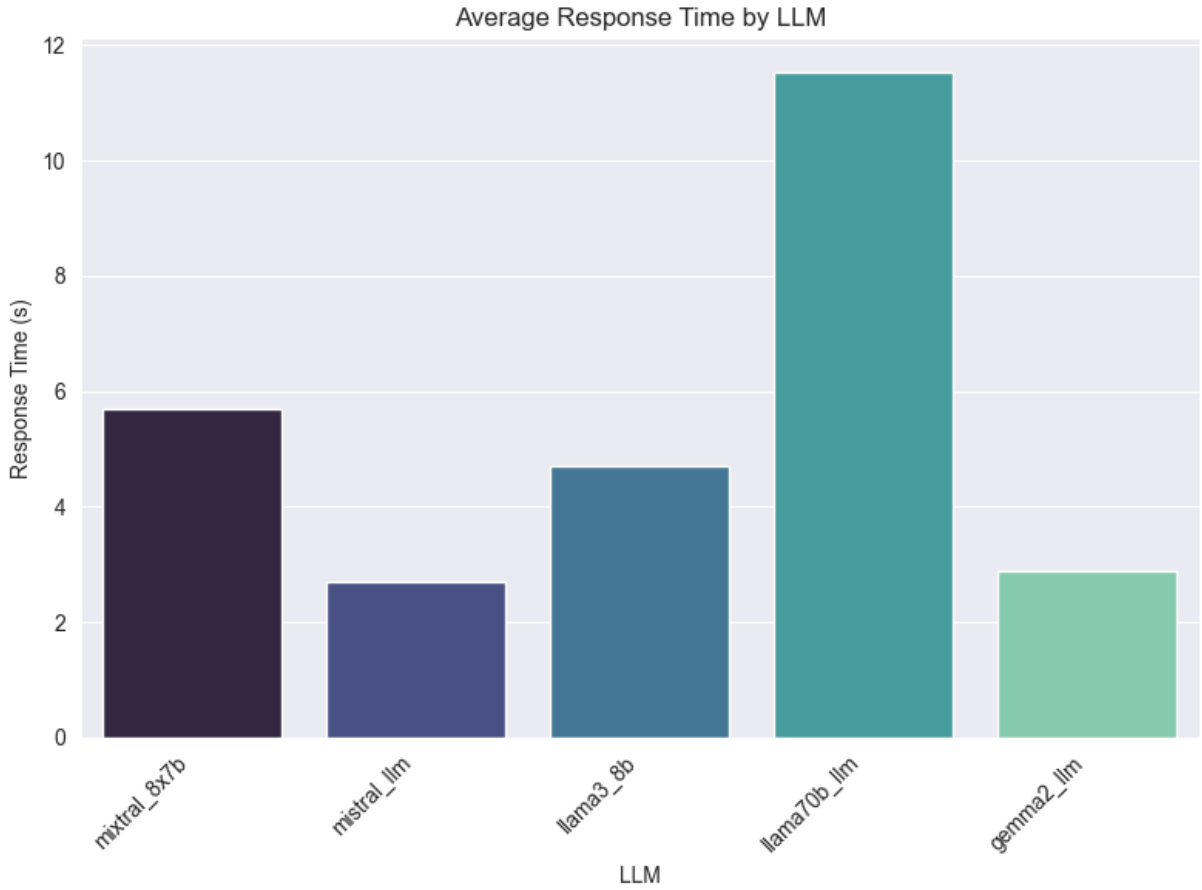


Figure 5.11: Bar plot illustrating the average response time (in seconds) for various LLMs, highlighting performance differences across models of varying architectures and sizes.

effective for tasks requiring a formal tone. Conversely, models with fewer parameters, like `Mistral-7B-v0.3`, struggled across all metrics, particularly with less optimized retrieval strategies.

* **Generation Time:** The response generation time varied significantly across models, with larger models such as `llama3:70b-instruct-q4_0` requiring over 11 seconds per response on average, reflecting their higher complexity. Lightweight models, such as `mistral_llm`, demonstrated faster generation times but at the expense of reduced correctness and faithfulness.

Conclusion: Efficient text extraction, high-performing embeddings, and robust retrieval strategies are pivotal to optimizing RAG systems. Larger models excel in correctness and faithfulness, while smaller models offer strengths in style consistency, highlighting trade-offs between computational efficiency and performance. The varying generation times further emphasize these trade-offs, with lightweight models being more suitable for real-time applications and larger models excelling in complex contextual reasoning tasks. Future work should focus on refining extraction techniques, optimizing retrieval strategies, and integrating advanced semantic evaluation metrics to further enhance the system, particularly for real-time and large-scale applications.

Chapter 6

Conclusions

This chapter recaps the work conducted throughout the study and addresses the limitations encountered and considers potential directions for future research.

In this thesis, we demonstrated how embeddings and semantic search enhance corporate document management systems by enabling efficient retrieval and chat-based assistance tailored to business needs. A systematic pipeline was central to this: extracting text from PDF documents, generating structured chunks, and storing embeddings in a ChromaDB database. The "bge" embedding model provided superior retrieval performance, validated by Mean Reciprocal Rank (MRR) evaluations.

Experiments with three retrieval strategies—basic similarity search, semantic reranking, and a merged approach with Maximal Marginal Relevance (MMR)—showed significant performance improvements. Integrating these retrievers into a Retrieval-Augmented Generation (RAG) framework enabled context-aware response generation using various large language models (LLMs).

This thesis contributes a versatile framework for embedding-based document management, bridging gaps in retrieval and contextualization. Despite challenges in consistent text structuring, it lays a foundation for scalable, adaptable systems applicable to diverse business domains.

6.1 Limitations

While the contributions are notable, several limitations arose during the development process:

* **Text Extraction:** The extraction of content from PDFs lacked a universal method to handle tables, titles, and other structured elements consistently. This impacted the ability to generate highly reliable text outputs, especially when dealing with complex document layouts.

* **Chunking Strategies:** Efforts to chunk extracted text into semantically meaningful units proved challenging. While current methods provided functional outputs, the lack of sentence-boundary-based chunking limited the system's precision in certain cases.

* **Scalability:** Although the system performed well in controlled settings, scalability to handle vast repositories of diverse documents remains an area for improvement, particularly in terms of storage and retrieval speed.

6.2 Future Work

To address these limitations and extend the capabilities of the system, several avenues for future work are proposed:

* **Improved Text Extraction Methods:** Developing or integrating advanced tools to accurately identify and extract tables, titles, and other structured elements will enhance the initial data preprocessing step.
* **Enhanced Chunking Strategies:** Exploring dynamic chunking methods that align more closely with sentence boundaries or semantic units will improve the organization and coherence of stored data.
* **Integrating RAPTOR:** To address challenges such as low accuracy, scalability issues, and inefficient ranking, incorporating RAPTOR—a state-of-the-art retrieval framework [49]—can enhance retrieval accuracy, optimize scalability, and improve efficiency.
* **Metadata Keyword Search:** Expanding the retrieval system to incorporate metadata-based searches, such as project names or subject-level keywords, will add another layer of functionality and improve relevance for users.
* **User Interface and Platform Integration:** Developing an intuitive user interface and integrating the system into corporate platforms will make the technology more accessible and user-friendly, ensuring smoother adoption by businesses.

6.3 Final Remarks

This thesis has demonstrated the potential of embedding-based and retrieval-augmented approaches to enhance corporate document management systems. By combining advanced retrieval techniques, semantic search, and state-of-the-art LLMs, we have laid the groundwork for a more intelligent and adaptable document management framework.

While challenges remain, the proposed solutions and future enhancements offer a clear pathway for progress. The insights gained from this research underscore the transformative potential of AI in addressing real-world business needs, making corporate knowledge retrieval more efficient, accurate, and scalable. With continued advancements in AI technologies, this work sets a precedent for future innovations in the domain.

Bibliography

[1] Patrick S. H. Lewis et al. "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks". In: *CoRR* abs/2005.11401 (2020). arXiv: 2005.11401. URL: https://arxiv.org/abs/2005.11401.

[2] Simona Sternad Zabukovšek, Sandra Jordan, and Samo Bobek. "Managing Document Management Systems' Life Cycle in Relation to an Organization's Maturity for Digital Transformation". In: *Sustainability* 15.21 (2023). ISSN: 2071-1050. DOI: 10.3390/su152115212. URL: https://www.mdpi.com/2071-1050/15/21/15212.

[3] PyPDF2 Developers. *PyPDF2*. Available: https://github.com/py-pdf/pypdf2. 2023.

[4] PDFMiner Developers. *PDFMiner: Python PDF Parsing and Text Extraction*. Available: https://pdfminersix.readthedocs.io/. 2023.

[5] PDFPlumber Developers. *PDFPlumber: Extracting Tables, Text, and Other Elements from PDF Documents*. Available: https://pdfplumber.readthedocs.io/. 2023.

[6] Artifex Software, Inc. *PyMuPDF Documentation*. Available: https://pymupdf.readthedocs.io/. 2023.

[7] Tomas Mikolov et al. *Efficient Estimation of Word Representations in Vector Space*. 2013. arXiv: 1301.3781 [cs.CL]. URL: https://arxiv.org/abs/1301.3781.

[8] Jeffrey Pennington, Richard Socher, and Christopher Manning. "GloVe: Global Vectors for Word Representation". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Ed. by Alessandro Moschitti, Bo Pang, and Walter Daelemans. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1532–1543. DOI: 10.3115/v1/D14-1162. URL: https://aclanthology.org/D14-1162.

[9] Piotr Bojanowski et al. "Enriching Word Vectors with Subword Information". In: *CoRR* abs/1607.04606 (2016). arXiv: 1607.04606. URL: http://arxiv.org/abs/1607.04606.

[10] Jacob Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *CoRR* abs/1810.04805 (2018). arXiv: 1810.04805. URL: http://arxiv.org/abs/1810.04805.

[11] Nils Reimers and Iryna Gurevych. "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks". In: *CoRR* abs/1908.10084 (2019). arXiv: 1908.10084. URL: http://arxiv.org/abs/1908.10084.

[12] Model Developers. *mxbai-embed-large*. Available at https://huggingface.co/mxbai-embed-large. 2024.

[13] Nomic AI. *nomic-embed-text*. Available at https://docs.nomic.ai/models/embed-text. 2024.

[14] Model Developers. *bge-m3-f16*. Available at https://huggingface.co/bge-m3-f16. 2024.

[15] Model Developers. *multilingual-e5-large*. Available at https://huggingface.co/intfloat/multilingual-e5-large. 2024.

[16] T.Y. Liu. *Learning to Rank for Information Retrieval*. Foundations and Trends® in Information Retrieval Series. Now Publishers, 2009. ISBN: 9781601982445. URL: https://books.google.it/books?id=5iZrhRkmvbQC.

[17] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval: The Concepts and Technology Behind Search*. Addison Wesley, 2011. ISBN: 9780321416919. URL: https://books.google.it/books?id=HbyAAAAACAAJ.

[18] Kalervo Järvelin and Jaana Kekäläinen. "IR evaluation methods for retrieving highly relevant documents". In: *Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '00. Athens, Greece: Association for Computing Machinery, 2000, 41–48. ISBN: 1581132263. DOI: 10.1145/345508.345545. URL: https://doi.org/10.1145/345508.345545.

[19] Beijing Academy of Artificial Intelligence. *BAAI/bge-reranker-v2-m3*. Available at https://huggingface.co/BAAI/bge-reranker-v2-m3. 2024.

[20] Rodrigo Frassetto Nogueira and Kyunghyun Cho. "Passage Re-ranking with BERT". In: *CoRR* abs/1901.04085 (2019). arXiv: 1901.04085. URL: http://arxiv.org/abs/1901.04085.

[21] Rodrigo Frassetto Nogueira et al. "Multi-Stage Document Ranking with BERT". In: *CoRR* abs/1910.14424 (2019). arXiv: 1910.14424. URL: http://arxiv.org/abs/1910.14424.

[22] Omar Khattab and Matei Zaharia. "ColBERT: Efficient and Effective Passage Search via Contextualized Late Interaction over BERT". In: *CoRR* abs/2004.12832 (2020). arXiv: 2004.12832. URL: https://arxiv.org/abs/2004.12832.

[23] Rodrigo Frassetto Nogueira, Zhiying Jiang, and Jimmy Lin. "Document Ranking with a Pretrained Sequence-to-Sequence Model". In: *CoRR* abs/2003.06713 (2020). arXiv: 2003.06713. URL: https://arxiv.org/abs/2003.06713.

[24] Ashish Vaswani et al. "Attention Is All You Need". In: *CoRR* abs/1706.03762 (2017). arXiv: 1706.03762. URL: http://arxiv.org/abs/1706.03762.

[25] OpenAI et al. *GPT-4 Technical Report*. 2024. arXiv: 2303.08774 [cs.CL]. URL: https://arxiv.org/abs/2303.08774.

[26] Hugo Touvron et al. *LLaMA: Open and Efficient Foundation Language Models*. 2023. arXiv: 2302.13971 [cs.CL]. URL: https://arxiv.org/abs/2302.13971.

[27] Mistral AI. *Mixtral: Sparse Mixture of Experts Models*. 2024. URL: https://mistral.ai/news/mixtral-8x22b/.

[28] Chuhan Wu et al. "Attentive Pooling with Learnable Norms for Text Representation". In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Ed. by Dan Jurafsky et al. Online: Association for Computational Linguistics, July 2020, pp. 2961–2970. DOI: 10.18653/v1/2020.acl-main.267. URL: https://aclanthology.org/2020.acl-main.267.

[29] Mistral AI. *Mixtral 8x22B Instruct Model*. Available at https://mistral.ai/news/mixtral-8x22b/. 2024.

[30] Mistral AI. *Mixtral 8x7B Model*. Available at https://arxiv.org/abs/2403.08295. 2024.

[31] Mistral AI. *Mistral-B-v0.3 Model*. Available at https://futureskillsacademy.com/blog/mistral-vs-mixtral/. 2024.

[32] Meta AI. *LLaMA 3 8B Instruct FP16 Model*. Available at https://huggingface.co/mistralai/Mixtral-8x22B-v0.1. 2024.

[33] Meta AI. *LLaMA 3 70B Instruct Q4_0 Model*. Available at https://huggingface.co/mistralai/Mixtral-8x22B-v0.1. 2024.

[34] Gemma AI. *Gemma2:9b Model*. Details unavailable as of publication. 2024.

[35] Jeff Johnson, Matthijs Douze, and Hervé Jégou. *Billion-scale similarity search with GPUs*. 2017. arXiv: 1702.08734 [cs.CV]. URL: https://arxiv.org/abs/1702.08734.

[36] Yu. A. Malkov and D. A. Yashunin. *Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs*. 2018. arXiv: 1603.09320 [cs.DS]. URL: https://arxiv.org/abs/1603.09320.

[37] ChromaDB Developers. *ChromaDB: Open-Source Vector Database for AI Applications*. Available at https://docs.trychroma.com. 2023.

[38] Vladimir Karpukhin et al. *Dense Passage Retrieval for Open-Domain Question Answering*. 2020. arXiv: 2004.04906 [cs.CL]. URL: https://arxiv.org/abs/2004.04906.

[39] Ellen Voorhees. *Overview of the TREC-9 Question Answering Track*. en. 2001. URL: https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=151529.

[40] Gerard Salton and Christopher Buckley. "Term-weighting approaches in automatic text retrieval". In: *Information Processing Management* 24.5 (1988), pp. 513–523. ISSN: 0306-4573. DOI: https://doi.org/10.1016/0306-4573(88)90021-0. URL: https://www.sciencedirect.com/science/article/pii/0306457388900210.

[41] David C. Blair and M. E. Maron. "An evaluation of retrieval effectiveness for a full-text document-retrieval system". In: *Commun. ACM* 28.3 (Mar. 1985), 289–299. ISSN: 0001-0782. DOI: 10.1145/3166.3197. URL: https://doi.org/10.1145/3166.3197.

[42] Jane Greenberg. "A quantitative categorical analysis of metadata elements in image-applicable metadata schemas". In: *J. Am. Soc. Inf. Sci. Technol.* 52.11 (Sept. 2001), 917–924. ISSN: 1532-2882. DOI: 10.1002/asi.1170.abs. URL: https://doi.org/10.1002/asi.1170.abs.

[43] S. E. Robertson. "The probability ranking principle in IR". In: *Readings in Information Retrieval*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1997, 281–286. ISBN: 1558604545.

[44] Marti A. Hearst. "Design recommendations for hierarchical faceted search interfaces". In: *ACM SIGIR Workshop on Faceted Search*. 2006.

[45] Carol Friedman and et al. "A knowledge-based system for retrieving and classifying information in a pathology report database". In: *Journal of the American Medical Informatics Association* 1.4 (1994), pp. 326–341.

[46] C.D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008. ISBN: 9781139472104. URL: https://books.google.it/books?id=t1PoSh4uwVcC.

[47] Raymond W. Smith. "An Overview of the Tesseract OCR Engine". In: *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)* 2 (2007), pp. 629–633. URL: https://api.semanticscholar.org/CorpusID:7038773.

[48] Nicole M. Radziwill and Morgan C. Benton. *Evaluating Quality of Chatbots and Intelligent Conversational Agents*. 2017. arXiv: 1704.04579 [cs.CY]. URL: https://arxiv.org/abs/1704.04579.

[49] Parth Sarthi et al. *RAPTOR: Recursive Abstractive Processing for Tree-Organized Retrieval*. 2024. arXiv: 2401.18059 [cs.CL]. URL: https://arxiv.org/abs/2401.18059.

[50] Relari AI. *Continuous Eval Documentation*. Version 0.3. 2024. URL: https://continuous-eval.docs.relari.ai/v0.3.

[51] Tom B. Brown et al. *Language Models are Few-Shot Learners*. 2020. arXiv: 2005.14165 [cs.CL]. URL: https://arxiv.org/abs/2005.14165.

[52] Daniel Cer et al. *Universal Sentence Encoder*. 2018. arXiv: 1803.11175 [cs.CL]. URL: https://arxiv.org/abs/1803.11175.