INTER-UNIVERSITY MASTER'S DEGREE PROGRAMME

Università degli Studi di Milano

Università degli Studi di Milano - Bicocca

Università di Pavia

Master's Degree Programme
Artificial Intelligence for Science and Technology

A Comprehensive Study on Retrieval Augmented Generation Methods for More Robust LLMs

Supervisor:
Dr. Napoletano Paolo

Candidate name:
Andrea Palmieri
Registration number:
921785

Academic Year 2024/2025

Here you can write whatever you want and here too.

Contents

Chapter 1

Introduction

The advent of Large Language Models (LLMs) has marked a pivotal moment in the field of artificial intelligence, demonstrating unprecedented capabilities in understanding and generating human-like text. These models, trained on vast and diverse datasets, have become the backbone of numerous applications, from sophisticated chatbots to content creation tools. However, their power is not without limitations. Two significant challenges inherent to LLMs are *knowledge cutoff* and *hallucination*. Knowledge cutoff refers to the fact that an LLM's knowledge is static and frozen at the point its training data was collected, rendering it unable to provide information on events or developments that occurred post-training. Hallucination is the tendency of these models to generate plausible but factually incorrect or nonsensical information, a byproduct of their probabilistic nature and their training objective to predict the next word rather than to state facts.

This thesis investigates Retrieval-Augmented Generation (RAG), a paradigm designed to directly address these shortcomings [1]. RAG enhances the capabilities of LLMs by grounding their responses in external, up-to-date knowledge sources. Instead of relying solely on its internal, parametric knowledge, a RAG system first retrieves relevant information from a specified corpus—such as a collection of scientific papers, a corporate knowledge base, or the entire web—and then uses this information to inform its generation process. This approach aims to mitigate hallucinations by providing factual context and overcomes the knowledge cutoff issue by allowing access to real-time information. The core principle is to combine the generative power of LLMs with the factual accuracy of information retrieval systems, leading to more robust, trustworthy, and contextually appropriate responses [2].

The RAG paradigm has evolved significantly since its inception. As categorized by Gao et al. (2024) [2], the landscape can be understood through three main stages:

- **Naive RAG:** The initial and most straightforward implementation, involving a simple retrieve-then-read process.
- **Advanced RAG:** Focuses on optimizing the retrieval stage through techniques like pre-retrieval processing and post-retrieval reranking.
- **Modular RAG:** A more flexible and powerful approach that treats RAG as a modular framework, allowing for greater adaptability and integration of different components and patterns.

This thesis will explore these concepts, starting with the foundational Naive RAG architecture and progressing to more advanced techniques. We will conduct a deep dive into the critical elements of the RAG pipeline, including:

- **Embedding Models:** The models used to convert text into numerical representations for semantic search.
- **Retrieval and Reranking:** The techniques for identifying the most relevant information and refining the selection, including reranking and dynamic thresholding.
- **Prompt Engineering:** The art of crafting effective prompts to guide the LLM in synthesizing the retrieved context.
- **LLM Selection:** The impact of choosing different generator models on the final output.

Through a series of structured experiments, this thesis will systematically evaluate how each of these components influences system performance, with the ultimate goal of providing a clear framework for building and optimizing robust RAG systems.

Chapter 2

Fundamentals of Retrieval-Augmented Generation

Retrieval-Augmented Generation (RAG) is an architectural pattern for LLM-based systems that combines a retrieval component with a generative model. This approach enhances the model's knowledge with external, up-to-date information, addressing common limitations like knowledge cutoffs and hallucination [1, 2]. This chapter lays the groundwork by explaining the core mechanics of RAG, its main challenges, and the role of its key components.

2.1 How it Works

The foundational RAG architecture, often referred to as **Naive RAG** [2], operates in two main stages: retrieval and generation, as originally proposed by Lewis et al. (2020) [1]. The process begins when a user submits a query. Instead of directly feeding the query to the LLM, the RAG pipeline intercepts it and first interacts with an external knowledge base.

1. **Retrieval Stage:** The user's query is converted into a numerical representation, or *embedding*, using a text embedding model. This query embedding is then used to search a pre-indexed collection of documents. The goal is to find and retrieve chunks of text that are semantically similar to the query. This retrieval is typically performed using a vector database, which is optimized for high-speed similarity searches over large datasets of embeddings. The output

of this stage is a set of relevant text chunks, often referred to as the *context*.

2. **Generation Stage:** The retrieved context is then combined with the original query into a structured prompt. This augmented prompt is fed to the LLM. By providing this explicit, relevant information, the LLM is guided to generate a response that is not only contextually appropriate but also grounded in the facts contained within the retrieved documents. This process significantly enhances the accuracy and factuality of the output.

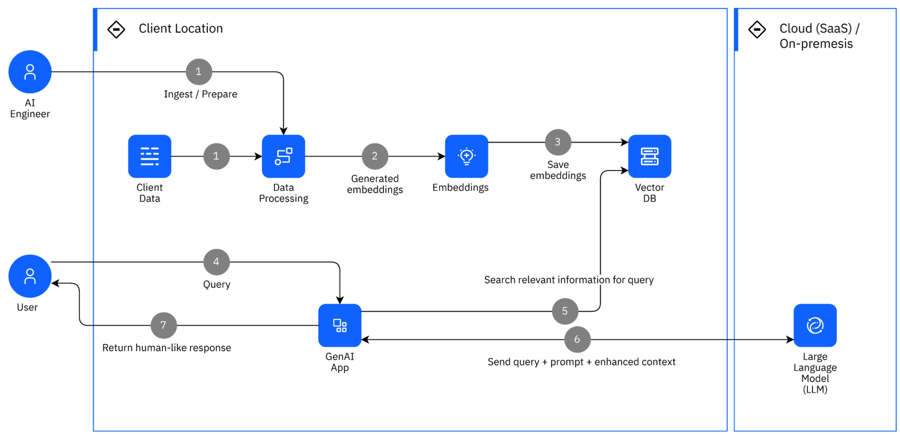


Figure 2.1: High-level architecture of a Retrieval-Augmented Generation system. Image from IBM [3].

2.2 Main Challenges

The apparent simplicity of the Naive RAG pipeline belies several complex challenges that must be addressed to build a robust and effective system [2].

2.2.1 Ensuring Content Relevance

The quality of the generated output is fundamentally dependent on the quality of the retrieved context. If the retriever fetches irrelevant or low-quality documents, the LLM may ignore them or, worse, incorporate incorrect information into its response. A key challenge is the *lost in the middle problem*, where LLMs tend to overlook relevant information if it is buried among less relevant chunks in the context window [4]. This phenomenon underscores the need for retrieval systems that not only find relevant documents but also rank them effectively.

2.2.2 Optimizing Retrieval vs. Generation Trade-offs

There is an inherent trade-off between the speed and comprehensiveness of the retrieval step. Retrieving more documents might increase the chance of finding the correct information but also increases the computational load and the risk of introducing noise. The length of the context that can be passed to the LLM is also limited by its context window size. As highlighted by Gao et al. (2024) [2], optimizing the selection of the most relevant document chunks from the initial similarity search is a key challenge. Techniques like re-ranking retrieved results, which will be explored later in this thesis, are designed to address this trade-off.

2.2.3 Handling Noisy or Conflicting Information

Real-world data is often messy. The retrieved context may contain conflicting facts or irrelevant details. The RAG system must be resilient to such noise, and the LLM must be capable of synthesizing information from multiple sources, identifying contradictions, and prioritizing the most reliable data. Advanced RAG architectures, such as those employing query transformations or reranking, are designed to tackle this issue [2, 5].

2.2.4 Seamless Integration and Synthesis

The LLM must be able to seamlessly weave the retrieved information into a coherent and natural-sounding response. This requires not just extracting facts but understanding the nuances of the context and integrating them into a cohesive narrative that directly answers the user's query. The fluency and relevance of the final output are direct measures of the RAG system's success.

2.3 Vector Databases and Similarity Search

Vector databases are a cornerstone of modern RAG systems, serving as the indexed knowledge base. They work by storing text data as high-dimensional vectors known as *embeddings*. When a query is received, it is also converted into an embedding, and the database searches for the vectors in its index that are closest to the query vector.

2.3.1 Measuring Similarity

The most common way to measure the distance between two vectors in the context of RAG is **cosine similarity**, which measures the cosine of the angle between them. For two vectors, A and B, the cosine similarity is calculated as:

$$\text{Cosine Similarity} = \frac{A \cdot B}{\|A\|\|B\|} \tag{2.1}$$

Where $A \cdot B$ is the dot product of the two vectors, and $\|A\|$ and $\|B\|$ are their magnitudes. The result ranges from -1 (exactly opposite) to 1 (exactly the same). Another common metric is **Euclidean distance**, which is the straight-line distance between two points in the vector space:

$$\text{Euclidean Distance} = \sqrt{\sum_{i=1}^{n}(A_i - B_i)^2} \tag{2.2}$$

2.3.2 The Advantage of Normalized Vectors

For efficiency, it is a common practice to **normalize** the vectors before storing them in the database. A normalized vector has a magnitude (or L2 norm) of 1. The magnitude of a vector $A = [a_1, a_2, \ldots, a_n]$ is computed as:

$$\|A\| = \sqrt{a_1^2 + a_2^2 + \cdots + a_n^2} \tag{2.3}$$

When vectors are normalized, the denominator in the cosine similarity formula ($\|A\|\|B\|$) becomes 1. Therefore, the cosine similarity calculation simplifies to just the **dot product** of the vectors, which is computationally much cheaper:

$$\text{Cosine Similarity (normalized)} = A \cdot B \tag{2.4}$$

This optimization avoids the expensive square root operations needed to calculate the vector magnitudes, allowing for significantly faster similarity searches, which is critical for real-time RAG applications.

2.3.3 Indexing for Efficient Search

To avoid a brute-force search, vector databases use specialized indexing algorithms for Approximate Nearest Neighbor (ANN) search. These algorithms, such as Hierarchical Navigable Small World (HNSW) [6] and Inverted File (IVF) [7], enable efficient retrieval of the top-k most similar vectors without having to compare the query vector to every single vector in the database. This is crucial for achieving low-latency responses in large-scale RAG systems.

2.4 Mitigating Hallucinations

One of the most significant benefits of RAG is its ability to mitigate LLM hallucinations. By providing factual, verifiable context directly within the prompt, RAG grounds the model's response in reality. The LLM is instructed to formulate its answer based on the provided text, reducing its reliance on its internal, parametric knowledge, which may be outdated or incorrect.

However, RAG is not a perfect solution. If the retrieved context is of poor quality, contains subtle inaccuracies, or is itself misleading, the LLM may still generate a flawed response. Therefore, the quality of the retrieval process is paramount. A well-tuned retriever that provides accurate and relevant context is the first and most critical line of defense against hallucinations in a RAG system [2].

Chapter 3

Retrieval and Optimization Methods

The retrieval component is the heart of a RAG system. Its ability to surface high-quality, relevant information from a vast corpus is the primary determinant of the system's overall performance. This chapter provides a detailed exploration of the critical techniques used to build and optimize the retrieval pipeline, from the initial processing of documents to the final reranking of retrieved candidates. We will follow the structure proposed by Gao et al. (2024) [2], which categorizes RAG optimization into pre-retrieval, retrieval, and post-retrieval stages.

3.1 Chunking Techniques

Chunking is a key pre-retrieval processing step that involves breaking down large documents into smaller, more manageable pieces. The goal is to create chunks that are semantically coherent and small enough to be efficiently processed by embedding models and fit within the context window of an LLM. The choice of chunking strategy has a profound impact on retrieval quality.

3.1.1 Naive vs. Semantic Chunking

Naive Chunking, also known as fixed-size chunking, is the simplest approach. It involves splitting documents into segments of a predetermined length (e.g., 200 words) with an optional overlap between adjacent chunks. While easy to implement, this method can be suboptimal as it often splits sentences or paragraphs in the middle, breaking the semantic continuity of the text.

Semantic Chunking represents a more sophisticated approach. Instead of relying on arbitrary lengths, it aims to divide the text at logical boundaries. This can be done in several ways:

- **Sentence-Level Chunking:** Using natural language processing libraries to split the text into individual sentences.
- **Recursive Chunking:** A hierarchical method that first tries to split by paragraphs, then by sentences, and finally by words, to maintain semantic coherence as much as possible.

3.2 Embedding Models

The choice of embedding model is critical for capturing the semantic meaning of the text. These models transform text into high-dimensional vectors, where semantically similar texts are located closer to each other in the vector space.

3.2.1 Contextual Embeddings

Modern RAG systems rely on **contextual embeddings**, such as those produced by transformer-based models like BERT, RoBERTa, and the OpenAI Ada series. Unlike older static word embeddings (e.g., Word2Vec, GloVe), which assign a single vector to each word, contextual embeddings generate a unique vector for a word based on the sentence it appears in. This allows them to capture nuances, ambiguity, and the richness of language, leading to more accurate semantic search.

3.2.2 Fine-tuning Embedding Models

For domain-specific applications, pre-trained embedding models may not perform optimally. Fine-tuning the embedding model on a dataset that is representative of the target domain can significantly improve retrieval relevance. This process adapts the model to the specific vocabulary and semantic relationships present in the corpus.

3.3 Post-Retrieval Reranking and Filtering

Once an initial set of documents is retrieved based on semantic similarity, their relevance and ordering can be further improved through post-retrieval processing. This is a key component of the **Advanced RAG** paradigm [2].

3.3.1 BM25 and TF-IDF for Reranking

Traditional information retrieval algorithms like **BM25** and **TF-IDF** are based on keyword matching. They are highly effective at finding documents that contain the exact keywords from the query. While dense retrievers (vector search) find what the user *means*, these sparse retrievers find what the user *says*. By using BM25 or TF-IDF to rerank the candidates retrieved by vector search, we can improve precision by boosting the rank of documents that have high lexical overlap with the query [2].

3.3.2 Hybrid Systems: Combining Similarity with BM25/TF-IDF

A fully **hybrid system** combines the scores from both dense (semantic) and sparse (keyword) retrieval methods from the outset. A common approach is to use a weighted combination of the scores from a vector search and a BM25 search to produce a final relevance score. This allows the system to leverage the strengths of both approaches, capturing both semantic relevance and keyword importance for a more robust retrieval process [2].

3.3.3 Cross-Encoder Rerankers

For the highest possible accuracy, a **cross-encoder** model can be used as a final reranking step. Unlike bi-encoders (standard embedding models) that create separate embeddings for the query and documents, a cross-encoder takes the query and a candidate document as a single input. This allows the model to perform a deep, token-by-token comparison, resulting in a highly accurate relevance score [8]. However, cross-encoders are computationally expensive and are typically only used to rerank a small number of top candidates from a faster, initial retrieval stage.

3.4 Dynamic Similarity Thresholding

Instead of retrieving a fixed number of chunks (top-N), **dynamic similarity thresholding** adapts the retrieval process based on the query itself. For some queries, only a few highly relevant chunks may be needed, while for others, a broader context is beneficial. Dynamic thresholding methods analyze the distribution of similarity scores for a given query and attempt to find a natural cutoff point, helping to retrieve a more contextually appropriate number of chunks. This prevents the inclusion of irrelevant documents when the similarity scores drop off sharply and allows for more comprehensive retrieval when many documents are similarly relevant.

3.5 Late Chunking with Contextual Chunk Embeddings

Late chunking is an advanced strategy that fundamentally changes how chunk embeddings are generated, moving away from the isolated processing of chunks to a more holistic, context-aware approach. As detailed by Günther et al. (2025) [9], this technique leverages the full power of long-context embedding models to create what they term *Contextual Chunk Embeddings*.

3.5.1 Limitations of Traditional Chunking

In a traditional chunking workflow, a document is first split into discrete chunks (e.g., by paragraphs or fixed token counts or for each page or by a different chunking strategy). Then, an embedding model is applied to each chunk independently to generate its vector representation. The major drawback of this method is context loss. The embedding for each chunk is created in a vacuum, unaware of the preceding or succeeding information in the original document. This can lead to ambiguous or less meaningful embeddings, thereby degrading the quality of the retrieval process, as the model cannot capture the full semantic richness of the text.

3.5.2 The Late Chunking Process

Late chunking addresses this limitation by reversing the process: it first generates highly contextualized embeddings at the token level and only then applies chunk boundaries. The process, detailed in Algorithm 1, is as follows:

1. **Tokenization and Contextualization:** Instead of chunking the document first, the entire document (or the largest possible segment that fits into the model's context window) is tokenized. The transformer model then processes this long sequence of tokens, generating a vector representation (ω_i) for every single token. Crucially, each of these token embeddings is context-aware, as it was created with knowledge of the entire surrounding text.

2. **Boundary Cue Application:** After the token-level embeddings ($\omega_1, \ldots, \omega_m$) have been generated, the predefined chunk boundaries are used. These boundaries, which are determined by a standard chunking algorithm (e.g., sentence

splitting), are not used to split the text for the model, but rather to identify which token embeddings correspond to which chunk. This is the key step from which the technique derives its name: the chunking logic is applied *late* in the process.

3. **Pooling:** With the token embeddings for each chunk identified, a pooling function—typically mean pooling—is applied to the sequence of token vectors within each chunk's boundaries. This aggregates the contextualized token embeddings into a single, final vector for each chunk.

This "inside-out" approach ensures that the final embedding for each chunk is not just a representation of the text within it, but is deeply informed by the broader context of the entire document, leading to more robust and accurate retrieval.

The concept of late chunking was introduced by Jina AI with the release of their `jina-embeddings-v2` model family. It has since been refined and extended in subsequent releases, including `jina-embeddings-v3` [10] and `jina-embeddings-v4` [11]. While the later versions introduced multimodal capabilities, which are not the focus of this study, the core principle of late chunking for text remains a key innovation.

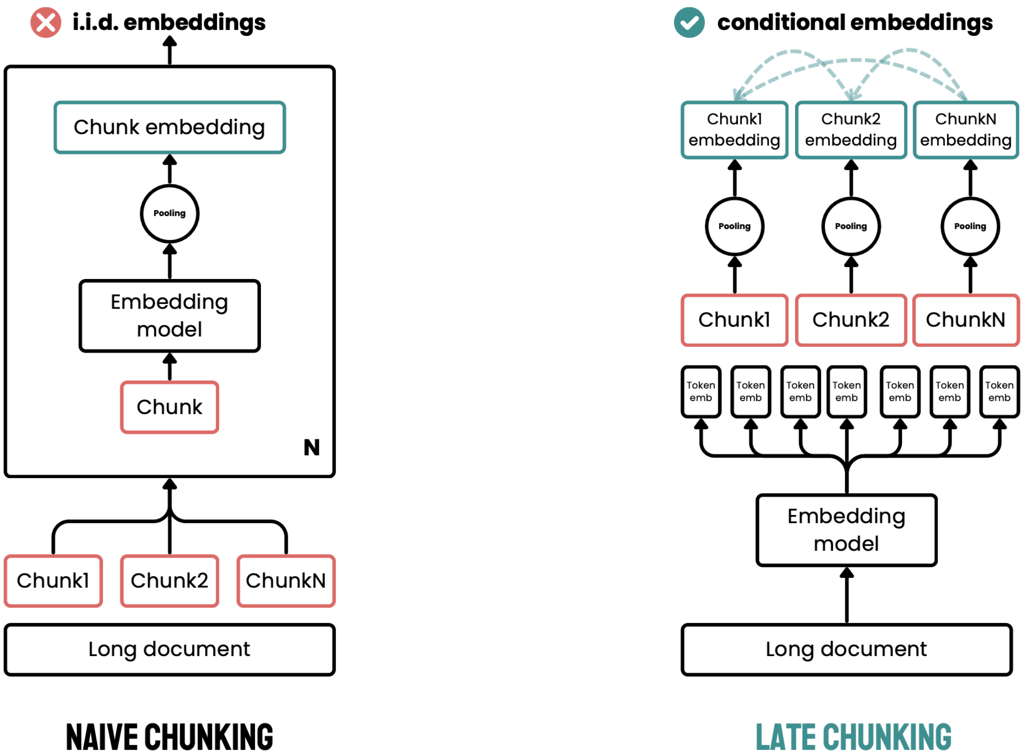


Figure 3.1: Visualization of the Late Chunking algorithm (right) compared to naive chunking (left). Image from Günther et al. (2025) [9].

Algorithm 1 Late Chunking

```
procedure LateChunking(document, chunk_boundaries)
    tokens ← tokenize(document)
    ω_1, ..., ω_m ← Transformer(tokens)        ▷ Generate token-level embeddings
    token_spans ← get_token_spans(document, tokens)
    chunk_token_indices ← []
    for each chunk in chunk_boundaries do
        start_char, end_char ← chunk
        start_token ← find_token_at_char(token_spans, start_char)
        end_token ← find_token_at_char(token_spans, end_char)
        append (start_token, end_token) to chunk_token_indices
    end for
    chunk_embeddings ← []
    for each start_idx, end_idx in chunk_token_indices do
        token_vectors_for_chunk ← ω_start_idx, ..., ω_end_idx
        embedding ← MeanPool(token_vectors_for_chunk)
        append embedding to chunk_embeddings
    end for
    return chunk_embeddings
end procedure
```

Chapter 4

Prompt Engineering for RAG

In a Retrieval-Augmented Generation system, the prompt is the critical bridge between the retrieved information and the generative capabilities of the Large Language Model. Effective prompt engineering is essential to guide the LLM in synthesizing the provided context and generating an accurate, relevant, and well-structured response. This chapter explores the fundamental principles of prompt design for RAG, different prompting styles, and the methodologies for evaluating their effectiveness.

4.1 Concepts of Prompt Engineering

A RAG prompt is more complex than a standard query to an LLM. It must effectively integrate two distinct pieces of information: the user's original query and the retrieved context. The structure of the prompt dictates how the LLM will perceive and use the provided information.

A typical RAG prompt includes:

- **Instructions:** Explicit directions to the LLM on how to behave. This might include instructions to use only the provided context, to adopt a certain persona, or to format the output in a specific way.
- **Context:** The set of retrieved document chunks that are relevant to the query.
- **Query:** The user's original question or request.

Crafting the prompt involves carefully arranging these elements. For example, a common template might look like this:

```
You are a helpful assistant. Use the following context
to answer the question at the end. If you don't know the
```

```
answer, just say that you don't know, don't try to make up
an answer.
   Context: [retrieved chunks]
   Question: [user query]
```

4.2 Prompt Styles

Different prompting styles can be employed depending on the specific task and the LLM being used. The choice of style can significantly influence the quality of the generated response.

- **Zero-Shot Prompting:** This is the most common style in RAG, where the LLM is given instructions and context but no prior examples of how to complete the task. The effectiveness of zero-shot prompts relies heavily on the clarity of the instructions and the LLM's inherent ability to reason and follow directions.

- **Few-Shot Prompting:** In this approach, the prompt includes a few examples (shots) of the desired input-output format. This can be particularly useful for complex tasks or when a very specific output structure is required. For RAG, this might involve showing the model how to synthesize context into an answer for a few sample questions before presenting the actual query.

- **Instruction-Based Prompts:** These prompts focus on providing very detailed and explicit instructions. This can include negative constraints (e.g., "Do not mention information not present in the context") and positive constraints (e.g., "Summarize the answer in three bullet points").

- **Role-Playing Prompts:** Assigning a role to the LLM (e.g., "You are a financial analyst reviewing these documents") can help to frame the context and guide the tone and focus of the response.

4.3 Prompt Evaluation

Evaluating the effectiveness of different prompts is a critical step in optimizing a RAG pipeline. Since the quality of a generated response can be subjective, a combination of qualitative and quantitative methods is often used.

4.3.1 Golden Datasets

A **golden dataset** is a curated collection of queries paired with ideal, human-verified answers. By comparing the LLM's output for a given query to the golden answer, one can assess the quality of the prompt. While creating a golden dataset can be labor-intensive, it provides a strong benchmark for evaluation.

4.3.2 LLM-as-a-Judge

A more scalable approach to evaluation is to use another, often more powerful, LLM as an evaluator, an approach that has been shown to correlate well with human judgment [12]. This is known as the **LLM-as-a-Judge** method. Frameworks like **DeepEval** have emerged to standardize this process. An evaluator LLM is given a set of criteria and asked to score the output of the RAG system on various dimensions.

4.4 Evaluation Metrics

When using an LLM-as-a-Judge, several key metrics are used to assess the quality of the RAG output. These metrics provide a multi-faceted view of performance:

- **G-Eval / DAG (Detail, Accuracy, Groundedness):** A composite score where the evaluator LLM assesses the response based on its level of detail, factual accuracy, and how well it is grounded in the provided context.

- **Answer Relevancy:** Measures how well the generated answer addresses the user's actual query. A factually correct answer is not useful if it does not answer the question.

- **Faithfulness:** This metric assesses whether the LLM's response is a faithful representation of the information in the retrieved context. A high faithfulness score means the model did not invent information that wasn't present in the source documents.

- **Hallucination Rate:** Specifically measures the presence of factually incorrect or nonsensical statements in the output. This is a critical metric for building trustworthy RAG systems.

By systematically testing different prompt structures and styles and evaluating them against these metrics, it is possible to identify the optimal prompt design that maximizes the performance of the RAG system for a given application.

Chapter 5

Comparative Analysis of Different LLMs

The Large Language Model serves as the generative component of the RAG system, responsible for synthesizing the retrieved information and generating the final response. The choice of the generator LLM is a critical decision that significantly impacts the system's overall performance, cost, and speed. This chapter provides a comparative analysis of different LLMs in the context of RAG, focusing on how their architectural differences, context window sizes, and inherent capabilities affect their suitability for the generation task.

5.1 The Role of the Generator LLM

In a RAG pipeline, the LLM's role is not to recall facts from its internal parametric memory but to reason over the provided context. As highlighted by Gao et al. (2024) [2], the ideal generator LLM should excel at:

- **Context Adherence (Faithfulness):** Strictly basing its answer on the provided context and avoiding the introduction of outside information.
- **Information Synthesis:** Weaving together facts from multiple retrieved chunks into a single, coherent answer.
- **Instruction Following:** Accurately following the instructions in the prompt, such as adhering to a specific output format or persona.
- **Noise Resistance:** Ignoring irrelevant or contradictory information within the context and focusing on the most relevant facts.

5.2 Performance Evaluation based on Model and Context Window Size

Different LLMs exhibit varying levels of proficiency in these areas. The performance of a RAG system is therefore highly dependent on the specific model chosen for the generation step. This section evaluates various models based on these criteria.

5.2.1 Comparing LLM Families

We can categorize LLMs into several families, each with its own characteristics. Our experiments evaluated models from the following prominent families:

- **GPT (Generative Pre-trained Transformer) Series (e.g., GPT-4, GPT-4o):** Developed by OpenAI, these models are known for their strong reasoning and instruction-following capabilities. GPT-4, in particular, has shown a high degree of faithfulness and an ability to synthesize complex information, making it a popular choice for high-quality RAG systems [13].

- **Llama Series (e.g., Llama 2, Llama 3):** Developed by Meta, these are powerful open-source models that have become highly competitive with their proprietary counterparts. They offer a strong balance of performance and customizability, allowing for fine-tuning on specific domains. While not evaluated in our final experiments, the Llama series represents a critical open-source alternative and a benchmark for performance in the field [14].

- **Claude Series (e.g., Claude 3 Sonnet, Opus):** Developed by Anthropic, these models are particularly noted for their large context windows and strong performance on tasks requiring complex reasoning and a deep understanding of long documents
citeanthropic2024claude3.

- **Gemini Series (e.g., Gemini 1.5 Pro, Gemini 2.5 Pro):** Developed by Google, the Gemini models are inherently multimodal and designed for high performance across a wide range of tasks. They have demonstrated strong results in both retrieval and generation, making them a versatile choice for RAG systems [15].

5.2.2 The Impact of Context Window Size

The **context window size** of an LLM defines the maximum amount of text (prompt + retrieved context + generated response) that the model can handle at one time. This has several implications for RAG, as discussed by Gao et al. (2024) [2]:

- **Information Density:** A larger context window allows for more retrieved chunks to be passed to the LLM, potentially increasing the comprehensiveness of the answer. However, this also increases the risk of the "lost in the middle" problem, where the model may overlook relevant information buried in a long context [4].
- **Cost and Latency:** Processing larger contexts is more computationally expensive, leading to higher operational costs and slower response times.
- **Architectural Differences:** Newer models with very large context windows (e.g., Claude 3) are designed to be more effective at finding and using information within long documents, which can be a significant advantage for certain RAG applications.

5.3 Trade-offs in LLM Selection

Choosing the right LLM for a RAG system involves balancing several factors:

- **Performance:** The quality of the generated output, measured by metrics like faithfulness, relevancy, and accuracy.
- **Cost:** The financial cost per generated token, which can vary significantly between models.
- **Speed (Latency):** The time it takes for the model to generate a response.
- **Customizability:** The ability to fine-tune the model on domain-specific data, which is often easier with open-source models.

Ultimately, the optimal choice depends on the specific requirements of the application. A customer-facing chatbot might prioritize speed and low cost, while a legal research assistant would prioritize the highest possible accuracy and faithfulness, even at a greater computational expense.

Chapter 6

Identifying Relevant Chunks for Responses

In a Retrieval-Augmented Generation system, the final response is a synthesis of information drawn from multiple retrieved document chunks. For the purposes of transparency, debuggability, and continuous improvement, it is crucial to understand exactly which pieces of the retrieved context were used to construct the answer. This chapter explores the methods and importance of analyzing the alignment between the generated response and the source chunks, a process often referred to as *citation and attribution* or *groundedness*.

6.1 The Importance of Chunk-Response Alignment

Understanding the link between the source context and the final output, as highlighted by Gao et al. (2024) [2], serves several key functions:

- **Trust and Verifiability:** For users, especially in critical applications like medical or legal research, being able to see the source of a particular statement is essential for trusting the system's output. Citations allow users to verify the information for themselves.

- **Debugging and Evaluation:** When a RAG system produces a suboptimal or incorrect answer, tracing the response back to the source chunks is the first step in diagnosing the problem. It helps to determine if the issue lies with the retriever (fetching irrelevant context), the generator (misinterpreting the context), or the source documents themselves.

- **System Improvement:** By analyzing which chunks are consistently used to answer certain types of questions, we can gain insights into the performance of the retrieval system. If high-quality chunks are being ignored or low-quality chunks are being used, it may indicate a need to fine-tune the embedding model or adjust the reranking strategy.

- **Feedback Loops:** In advanced RAG systems, identifying useful chunks can provide a feedback signal to the retriever, allowing it to learn and improve its performance over time through reinforcement learning or other adaptive methods.

6.2 Methods for Analyzing Chunk-Response Alignment

Several techniques can be used to trace the provenance of the information in the generated response. The complexity and accuracy of these methods can vary significantly, and they often involve a trade-off between computational cost and precision.

6.2.1 Prompt-Based Attribution

The simplest method is to explicitly ask the LLM to cite its sources in the prompt. The instructions might include a directive like: *"After each sentence in your response, cite the ID of the source document you used to formulate that sentence."*

While straightforward, this approach has limitations. The LLM may not always follow the instructions perfectly, and it can sometimes hallucinate citations or incorrectly attribute information. The reliability of this method is highly dependent on the instruction-following capabilities of the chosen LLM.

6.2.2 Post-Hoc Similarity Analysis

Another approach is to analyze the alignment after the response has been generated. This can be done by:

1. Breaking down the generated response into individual sentences or claims.

2. For each sentence, calculating its embedding.

3. Comparing the embedding of the generated sentence to the embeddings of the original retrieved chunks.

4. The chunk with the highest semantic similarity to a given sentence is considered its most likely source.

This method provides a more quantitative and verifiable way to link the output to the input, but it is not foolproof. A generated sentence might synthesize information from multiple chunks, making a one-to-one mapping difficult.

6.2.3 Analyzing Attention Mechanisms

For transformer-based LLMs, the internal *attention mechanism* can theoretically provide insights into which parts of the input context were most influential in generating a particular part of the output. By inspecting the attention weights, one could see which of the retrieved chunks the model was "paying attention to" when it generated a specific word or phrase.

In practice, this is a highly complex approach. Accessing and interpreting attention weights can be difficult, especially with proprietary, black-box models. Furthermore, the relationship between high attention scores and factual contribution is not always direct and is an ongoing area of research.

6.2.4 Building a Knowledge Graph

A more structured approach involves building a knowledge graph from the source documents. The graph would contain entities and their relationships. When a response is generated, the entities mentioned in the response can be linked back to the nodes in the knowledge graph, providing a clear and structured form of attribution. This is a powerful but resource-intensive method that is best suited for well-defined domains.

6.3 Evaluation Metrics for Retrieval

As discussed by Gao et al. (2024) [2], evaluating the retrieval component is crucial for understanding the overall RAG system performance. Key metrics include:

- **Precision@k:** The proportion of relevant documents among the top-k retrieved documents.

- **Recall@k:** The proportion of all relevant documents in the corpus that are found in the top-k retrieved documents.

- **Mean Reciprocal Rank (MRR):** The average of the reciprocal ranks of the first relevant document for a set of queries. A higher MRR indicates that the system is better at ranking relevant documents higher.
- **Normalized Discounted Cumulative Gain (nDCG):** A measure of ranking quality that considers the graded relevance of documents.

These metrics help quantify the effectiveness of the retrieval stage in isolation from the generation stage.

Chapter 7

Experiments and Results

This chapter details the experimental methodologies and presents the findings from a series of evaluations designed to assess various components and strategies within Retrieval-Augmented Generation systems. The experiments cover a range of techniques from fundamental retrieval and chunking methods to advanced reranking, prompt engineering, and the impact of different embedding and generative models. The overarching goal is to identify optimal configurations for robust and effective RAG pipelines.

7.1 General Experimental Setup

To ensure a systematic and reproducible evaluation, all experiments were conducted using a consistent setup.

7.1.1 Dataset(s)

The primary dataset used for these experiments is a curated collection of academic papers and articles related to the field of artificial intelligence. This dataset was chosen for its complexity, technical vocabulary, and the need for precise, fact-based answers. For question generation and evaluation, a set of 100 questions was manually crafted, covering a range of topics within the dataset. Each question is designed to have a verifiable answer within the document corpus.

7.1.2 Baseline Configuration

To measure the impact of different optimization techniques, a baseline RAG configuration was established:

- **Embedding Model:** OpenAI `text-embedding-ada-002`, a widely used and strong baseline model.
- **Chunking Strategy:** Naive fixed-size chunking with a chunk size of 300 tokens and an overlap of 50 tokens.
- **Retrieval Strategy:** Basic cosine similarity search with a fixed retrieval of the top 5 most similar chunks (Top-N).
- **Generator LLM:** `GPT-4`.
- **Prompt Template:** A standard zero-shot prompt instructing the LLM to answer the question based on the provided context.

7.1.3 Core Evaluation Metrics

The performance of the RAG system was evaluated at both the retrieval and generation stages.

- **Retrieval Performance:** Assessed using *Precision@k*, *Recall@k*, and *Mean Reciprocal Rank (MRR)*. These metrics are crucial for understanding the effectiveness of the retrieval stage in isolation.
- **Generation Quality:** Evaluated using the *LLM-as-a-Judge* method with the DeepEval framework. The key metrics tracked were *Faithfulness*, *Answer Relevancy*, and *Hallucination Rate* [12].

7.2 Experiment 1: Embedding Models

This experiment evaluates the performance of different embedding models on long-context retrieval tasks. The goal is to understand how different architectural and chunking strategies affect retrieval performance on documents of varying lengths and complexity.

7.2.1 Models and Methodology

We compared five embedding models on the **LongEmbed** benchmark [16], a suite of datasets designed to test long-context retrieval. The models evaluated were:

- **OpenAI `text-embedding-ada-002`:** A widely-used, high-performance proprietary model with a vector size of 1536. This model does not require any specific task instruction.

- **Nomic AI `nomic-embed-text`:** An open-source model with a vector size of 768. This model uses a prefix for different tasks. For our retrieval task, we used ‘search_query:‘ for queries and ‘search_document:‘ for documents.

- **Jina AI `jina-embeddings-v2-base-en`:** An open-source model with a vector size of 768. This model, like Ada-002, does not require a task instruction.

- **Jina AI `jina-embeddings-v3`:** A model that implements *late chunking*, where chunks from the same document are tokenized together up to the model’s context window of 8192 tokens, with a vector size of 1024. It requires a ‘task‘ argument, for which we used ‘retrieval.query‘ and ‘retrieval.passage‘ for queries and documents respectively.

- **Qwen `Qwen3-Embedding-0.6B`:** An open-source model with a vector size of 1024. This model benefits from a prompt for queries, for which we used the recommended ‘query‘ prompt name.

For vector storage and retrieval, we used Milvus [17], a popular open-source vector database. We created an index of type HNSW (Hierarchical Navigable Small World) with the L2 distance metric. The index parameters were set to M=8 and efConstruction=64.

7.2.2 Datasets

The LongEmbed benchmark is composed of six datasets, two of which are synthetic and four are based on real-world data. The datasets are:

- **2WikiMQA:** A multi-hop question-answering dataset.

- **NarrativeQA:** A question-answering dataset with long stories.

- **Needle In A Haystack (NIAH):** A synthetic dataset that tests the model’s ability to find a specific piece of information ("needle") in a long text ("haystack").

- **Passkey Retrieval:** A synthetic dataset that tests the model’s ability to retrieve a specific key from a long document.

- **QMSum:** A query-based meeting summarization dataset.

- **SummScreenFD:** A summarization dataset based on TV show scripts.

7.2.3 Results and Discussion

The retrieval performance of the five embedding models across the six datasets is detailed in Tables 7.1 through 7.4, with an aggregated summary in Table 7.5. The results not only highlight a clear winner within our long-context benchmark but also reflect a significant paradigm shift in the broader landscape of text embedding models.

The Rise of Compact, High-Performance Models While our results show **`Qwen3-Embedding-0.6B`** as the top performer, its true significance is revealed when contextualized within the global Massive Text Embedding Benchmark (MTEB) leaderboard. As of August 1st, 2025, this model ranks an impressive 4th overall, surpassed only by Google's proprietary `gemini-embedding-001` and its own, much larger 4B and 8B parameter siblings [18]. With only 595M parameters, its performance is remarkable, demonstrating that architectural innovations and advanced training pipelines can allow smaller, open-source models to outperform far larger competitors. This efficiency is a direct result of its foundation on the Qwen3 architecture [19] and a sophisticated multi-stage training process that uses the foundation LLM itself to synthesize high-quality training data [20].

A New Baseline: The Obsolescence of Ada-002 For years, OpenAI's **`ada-002`** has been a de facto industry standard. However, these results, combined with its MTEB ranking of 167th, confirm that it has been decisively overcome. Newer models leveraging innovative architectures and specialized training techniques now offer vastly superior performance. The other models in our test also place significantly higher than `ada-002` on the MTEB leaderboard: `jina-embeddings-v3` ranks 24th, `nomic-embed-text` ranks 54th, and even the older `jina-embeddings-v2-base-en` ranks 154th [18]. This trend signals that the benchmark for state-of-the-art has moved towards more open and efficient models.

The Enduring Value of Specialization Despite the dominance of a strong generalist model like Qwen3, our findings also underscore the continued importance of specialized models. **`jina-embeddings-v3`**, with its unique late chunking strategy, remains the top performer on the `passkey` task, proving the value of its architectural design for retrieving specific facts from structured long documents. Likewise, **`nomic-embed-text`**'s superior performance on the `needle` dataset confirms its status as a premier model for extreme "needle-in-a-haystack" scenarios. This demonstrates that for specific, challenging use cases, a specialized model can still be the optimal choice.

Distance Metrics and Model Compatibility For most models, the choice between L2 and Inner Product (IP) distance metrics yields comparable results, indicating that their embeddings are well-normalized. A critical observation, however, is the performance degradation of **`jina-embeddings-v2-base-en`** when using the IP metric. As seen across all tables, its performance collapses with IP; for example, its average Recall@5 drops from 51.85% (L2) to a mere 12.12% (IP). This underscores that its embedding vectors are not normalized for cosine similarity, making L2 distance the only viable choice for this model and highlighting the importance of aligning the retrieval metric with the model's output properties.

Conclusion In summary, this experiment illustrates a pivotal moment in the evolution of text embedding models. Our findings lead to the following conclusions:

- The embedding landscape is no longer dominated by large, proprietary models. The exceptional performance of **`Qwen3-Embedding-0.6B`**, a compact 595M parameter model, signals a shift towards more efficient, accessible, and open-source solutions that achieve state-of-the-art results.
- Models like **`ada-002`**, while historically important, are now outdated. They have been surpassed by a new generation of models featuring more advanced architectures, training methodologies, and specialized strategies like late chunking.
- While general-purpose models have improved dramatically, **specialization remains highly valuable**. For niche but critical tasks like extreme long-context fact retrieval, models like `jina-embeddings-v3` and `nomic-embed-text` provide targeted performance that can exceed even the best generalists.
- **Technical implementation details are critical**. The choice of distance metric must be compatible with the model's properties, as the failure of `jina-embeddings-v2-base-en` with the IP metric demonstrates.

Table 7.1: Detailed RECALL Performance (in %). Higher is better. Results are grouped by k-value. Best result per column within each k-value group is in **bold**.

k	Model	2wikimqa		narrativeqa		needle		passkey		qmsum		summ screen fd		Average	
		L2	IP	L2	IP	L2	IP	L2	IP	L2	IP	L2	IP	L2	IP
1	Ada-002	79.33	81.00	54.44	54.27	2.00	1.50	**13.25**	**14.50**	45.65	45.91	86.90	86.90	52.01	51.98
	Jina-V2	**91.67**	24.67	41.43	2.32	3.75	0.00	12.75	8.00	40.54	11.07	81.55	32.14	41.48	4.66
	Jina-V3	90.67	**90.67**	57.54	57.46	1.25	0.75	10.50	12.50	47.22	47.22	84.23	84.23	54.69	54.67
	Nomic	67.33	67.33	47.51	48.15	**8.75**	**9.25**	5.75	3.75	21.02	21.02	52.98	52.98	42.67	43.13
	Qwen3-0.6B	89.67	89.67	**62.99**	**62.85**	4.25	3.75	9.50	13.75	**50.49**	**50.49**	**88.10**	**88.10**	**59.45**	**59.45**
5	Ada-002	87.33	88.00	63.85	63.64	4.25	4.00	32.50	34.00	65.82	66.34	96.73	96.73	62.71	62.66
	Jina-V2	95.67	66.00	50.31	6.02	5.75	0.00	29.50	15.25	62.08	32.87	95.54	69.94	51.85	12.12
	Jina-V3	**96.67**	**96.67**	67.40	67.42	2.00	2.00	**40.50**	**44.50**	67.58	67.58	97.92	97.92	66.09	66.22
	Nomic	79.67	79.67	58.64	59.33	**14.50**	**14.75**	9.25	5.25	39.03	39.03	82.74	82.74	54.69	55.11
	Qwen3-0.6B	95.67	95.67	**73.27**	**73.16**	5.50	6.50	15.75	40.75	**72.30**	**72.30**	**98.21**	**98.21**	**70.55**	**71.23**
10	Ada-002	90.00	90.67	66.35	66.22	5.25	5.00	38.75	41.00	73.94	74.46	**98.81**	**98.81**	65.91	65.94
	Jina-V2	97.67	77.67	53.39	7.71	6.25	0.25	31.25	16.50	72.30	45.12	97.62	81.85	55.58	15.43
	Jina-V3	**98.33**	**98.33**	70.67	70.71	2.25	2.50	**53.50**	**58.75**	75.05	75.05	**98.81**	**98.81**	69.94	70.13
	Nomic	85.00	85.00	62.32	62.88	**17.25**	**17.50**	9.25	5.25	47.35	47.35	91.07	91.07	58.92	59.24
	Qwen3-0.6B	98.00	98.00	**76.48**	**76.18**	6.25	7.00	16.00	42.25	**79.96**	**79.96**	98.21	98.21	**74.00**	**74.58**
25	Ada-002	92.33	92.67	69.44	69.45	7.50	7.50	49.75	53.25	82.25	82.91	**99.70**	**99.70**	69.74	69.93
	Jina-V2	98.67	91.67	56.97	11.65	8.50	1.50	33.25	17.75	81.99	64.37	97.92	93.45	59.63	21.37
	Jina-V3	**99.33**	**99.33**	74.30	74.28	5.25	5.25	**54.25**	**60.00**	83.76	83.76	**99.70**	**99.70**	73.92	74.08
	Nomic	89.33	89.33	66.78	67.31	**20.00**	**19.75**	9.25	5.25	62.41	62.41	96.13	96.13	64.41	64.70
	Qwen3-0.6B	98.33	98.33	**79.88**	**79.71**	8.25	11.25	16.00	42.25	**89.00**	**89.00**	**99.70**	**99.70**	**77.79**	**78.53**

Table 7.2: Detailed PRECISION Performance (in %). Higher is better. Results are grouped by k-value. Best result per column within each k-value group is in **bold**.

k	Model	2wikimqa		narrativeqa		needle		passkey		qmsum		summ screen fd		Average	
		L2	IP	L2	IP	L2	IP	L2	IP	L2	IP	L2	IP	L2	IP
1	Ada-002	79.33	81.00	54.44	54.27	2.00	1.50	**13.25**	**14.50**	45.65	45.91	86.90	86.90	52.01	51.98
	Jina-V2	**91.67**	24.67	41.43	2.32	3.75	0.00	12.75	8.00	40.54	11.07	81.55	32.14	41.48	4.66
	Jina-V3	90.67	**90.67**	57.54	57.46	1.25	0.75	10.50	12.50	47.22	47.22	84.23	84.23	54.69	54.67
	Nomic	67.33	67.33	47.51	48.15	**8.75**	**9.25**	5.75	3.75	21.02	21.02	52.98	52.98	42.67	43.13
	Qwen3-0.6B	89.67	89.67	**62.99**	**62.85**	4.25	3.75	9.50	13.75	**50.49**	**50.49**	**88.10**	**88.10**	**59.45**	**59.45**
5	Ada-002	17.47	17.60	12.77	12.73	0.85	0.80	6.50	6.80	13.16	13.27	19.35	19.35	12.54	12.53
	Jina-V2	19.13	13.20	10.06	1.20	1.15	0.00	5.90	3.05	12.42	6.57	19.11	13.99	10.37	2.42
	Jina-V3	**19.33**	**19.33**	13.48	13.48	0.40	0.40	**8.10**	**8.90**	13.52	13.52	19.58	19.58	13.22	13.24
	Nomic	15.93	15.93	11.73	11.87	**2.90**	**2.95**	1.85	1.05	7.81	7.81	16.55	16.55	10.94	11.02
	Qwen3-0.6B	19.13	19.13	**14.65**	**14.63**	1.10	1.30	3.15	8.15	**14.46**	**14.46**	**19.64**	**19.64**	**14.11**	**14.25**
10	Ada-002	9.00	9.07	6.64	6.62	0.53	0.50	3.87	4.10	7.39	7.45	**9.88**	**9.88**	6.59	6.59
	Jina-V2	9.77	7.77	5.34	0.77	0.63	0.03	3.12	1.65	7.23	4.51	9.76	8.18	5.56	1.54
	Jina-V3	**9.83**	**9.83**	7.07	7.07	0.22	0.25	**5.35**	**5.88**	7.50	7.50	**9.88**	**9.88**	6.99	7.01
	Nomic	8.50	8.50	6.23	6.29	**1.72**	**1.75**	0.93	0.53	4.73	4.73	9.11	9.11	5.89	5.92
	Qwen3-0.6B	9.80	9.80	**7.65**	**7.62**	0.63	0.70	1.60	4.22	**8.00**	**8.00**	9.82	9.82	**7.40**	**7.46**
25	Ada-002	3.69	3.71	2.78	2.78	0.30	0.30	1.99	2.13	3.29	3.32	**3.99**	**3.99**	2.79	2.80
	Jina-V2	3.95	3.67	2.28	0.47	0.34	0.06	1.33	0.71	3.28	2.57	3.92	3.74	2.39	0.85
	Jina-V3	**3.97**	**3.97**	2.97	2.97	0.21	0.21	**2.17**	**2.40**	3.35	3.35	**3.99**	**3.99**	2.96	2.96
	Nomic	3.57	3.57	2.67	2.69	**0.80**	**0.79**	0.37	0.21	2.50	2.50	3.85	3.85	2.58	2.59
	Qwen3-0.6B	3.93	3.93	**3.20**	**3.19**	0.33	0.45	0.64	1.69	**3.56**	**3.56**	**3.99**	**3.99**	**3.11**	**3.14**

Table 7.3: Detailed MRR Performance (in %). Higher is better. Results are grouped by k-value. Best result per column within each k-value group is in **bold**.

k	Model	2wikimqa		narrativeqa		needle		passkey		qmsum		summ screen fd		Average	
		L2	IP	L2	IP	L2	IP	L2	IP	L2	IP	L2	IP	L2	IP
	Ada-002	79.33	81.00	54.44	54.27	2.00	1.50	**13.25**	**14.50**	45.65	45.91	86.90	86.90	52.01	51.98
	Jina-V2	**91.67**	24.67	41.43	2.32	3.75	0.00	12.75	8.00	40.54	11.07	81.55	32.14	41.48	4.66
1	Jina-V3	90.67	**90.67**	57.54	57.46	1.25	0.75	10.50	12.50	47.22	47.22	84.23	84.23	54.69	54.67
	Nomic	67.33	67.33	47.51	48.15	**8.75**	**9.25**	5.75	3.75	21.02	21.02	52.98	52.98	42.67	43.13
	Qwen3-0.6B	89.67	89.67	**62.99**	**62.85**	4.25	3.75	9.50	13.75	**50.49**	**50.49**	**88.10**	**88.10**	**59.45**	**59.45**
	Ada-002	82.28	83.57	58.14	57.90	2.81	2.49	20.36	21.56	53.30	53.62	91.11	91.11	56.17	56.08
	Jina-V2	93.13	39.76	44.87	3.62	4.55	0.00	19.76	10.88	48.47	18.67	87.17	46.11	45.47	7.31
5	Jina-V3	**93.22**	**93.22**	61.43	61.34	1.46	1.21	**20.45**	23.56	54.77	54.77	90.10	90.10	59.09	59.11
	Nomic	72.21	72.21	51.83	52.49	**11.28**	**11.63**	7.20	4.46	27.64	27.64	64.18	64.18	47.30	47.75
	Qwen3-0.6B	92.28	92.28	**67.02**	**66.88**	4.83	4.78	12.18	**24.09**	**58.59**	**58.59**	**92.31**	**92.31**	**63.77**	**64.02**
	Ada-002	82.68	83.96	58.48	58.25	2.94	2.61	21.22	22.54	54.40	54.73	91.39	91.39	56.61	56.53
	Jina-V2	93.42	41.35	45.29	3.84	4.63	0.03	20.01	11.04	49.84	20.31	87.45	47.74	45.97	7.76
10	Jina-V3	**93.44**	**93.44**	61.87	61.78	1.49	1.27	**22.38**	**25.70**	55.79	55.79	90.23	90.23	59.61	59.64
	Nomic	72.96	72.96	52.33	52.97	**11.66**	**12.01**	7.20	4.46	28.74	28.74	65.37	65.37	47.87	48.30
	Qwen3-0.6B	92.60	92.60	**67.45**	**67.29**	4.92	4.84	12.22	24.33	**59.65**	**59.65**	**92.31**	**92.31**	**64.24**	**64.47**
	Ada-002	82.83	84.08	58.68	58.46	3.07	2.75	21.91	23.29	54.93	55.27	91.44	91.44	56.86	56.79
	Jina-V2	93.48	42.27	45.52	4.08	4.75	0.10	20.12	11.12	50.45	21.52	87.47	48.52	46.23	8.12
25	Jina-V3	**93.52**	**93.52**	62.10	62.01	1.68	1.43	**22.42**	**25.78**	56.35	56.35	90.28	90.28	59.87	59.89
	Nomic	73.21	73.21	52.61	53.24	**11.86**	**12.17**	7.20	4.46	29.69	29.69	65.69	65.69	48.22	48.64
	Qwen3-0.6B	92.62	92.62	**67.67**	**67.52**	5.05	5.13	12.22	24.33	**60.21**	**60.21**	**92.41**	**92.41**	**64.48**	**64.73**

Table 7.4: Detailed NDCG Performance (in %). Higher is better. Results are grouped by k-value. Best result per column within each k-value group is in **bold**.

k	Model	2wikimqa		narrativeqa		needle		passkey		qmsum		summ screen fd		Average	
		L2	IP	L2	IP	L2	IP	L2	IP	L2	IP	L2	IP	L2	IP
1	Ada-002	79.33	81.00	54.44	54.27	2.00	1.50	**13.25**	**14.50**	45.65	45.91	86.90	86.90	52.01	51.98
	Jina-V2	**91.67**	24.67	41.43	2.32	3.75	0.00	12.75	8.00	40.54	11.07	81.55	32.14	41.48	4.66
	Jina-V3	90.67	**90.67**	57.54	57.46	1.25	0.75	10.50	12.50	47.22	47.22	84.23	84.23	54.69	54.67
	Nomic	67.33	67.33	47.51	48.15	**8.75**	**9.25**	5.75	3.75	21.02	21.02	52.98	52.98	42.67	43.13
	Qwen3-0.6B	89.67	89.67	**62.99**	**62.85**	4.25	3.75	9.50	13.75	**50.49**	**50.49**	**88.10**	**88.10**	**59.45**	**59.45**
5	Ada-002	83.53	84.68	59.57	59.34	3.16	2.87	23.38	24.65	56.42	56.79	92.54	92.54	57.81	57.73
	Jina-V2	93.76	46.27	46.23	4.21	4.85	0.00	22.23	11.98	51.86	22.19	89.28	52.05	47.07	8.50
	Jina-V3	**94.09**	**94.09**	62.93	62.87	1.59	1.41	**25.38**	**28.73**	57.96	57.96	92.10	92.10	60.84	60.89
	Nomic	74.07	74.07	53.54	54.21	**12.10**	**12.43**	7.72	4.66	30.48	30.48	68.81	68.81	49.15	49.59
	Qwen3-0.6B	93.14	93.14	**68.59**	**68.46**	5.01	5.21	13.09	28.26	**62.01**	**62.01**	**93.80**	**93.80**	**65.47**	**65.82**
10	Ada-002	84.45	85.57	60.39	60.18	3.48	3.18	25.43	26.96	59.07	59.44	93.22	93.22	58.86	58.80
	Jina-V2	94.43	50.08	47.24	4.76	5.03	0.07	22.82	12.37	55.17	26.15	89.95	55.95	48.28	9.58
	Jina-V3	**94.63**	**94.63**	63.99	63.93	1.67	1.56	**29.77**	**33.58**	60.40	60.40	92.39	92.39	62.10	62.16
	Nomic	75.84	75.84	54.74	55.35	**13.01**	**13.32**	7.72	4.66	33.15	33.15	71.58	71.58	50.53	50.93
	Qwen3-0.6B	93.91	93.91	**69.63**	**69.44**	5.23	5.37	13.17	28.79	**64.52**	**64.52**	**93.80**	**93.80**	**66.59**	**66.91**
25	Ada-002	85.01	86.06	61.15	60.97	4.02	3.77	28.11	29.93	61.11	61.52	93.43	93.43	59.80	59.78
	Jina-V2	94.67	53.55	48.11	5.70	5.55	0.37	23.28	12.69	57.53	30.86	90.03	58.85	49.27	11.02
	Jina-V3	**94.90**	**94.90**	64.89	64.81	2.40	2.21	**29.95**	**33.88**	62.54	62.54	92.61	92.61	63.08	63.13
	Nomic	76.88	76.88	55.83	56.43	**13.71**	**13.90**	7.72	4.66	36.83	36.83	72.83	72.83	51.87	52.26
	Qwen3-0.6B	93.99	93.99	**70.47**	**70.31**	5.73	6.43	13.17	28.79	**66.71**	**66.71**	**94.18**	**94.18**	**67.52**	**67.89**

Table 7.5: Summary of Average Performance Across All Datasets (in %). Best result per column within each k-group is in **bold**.

k	Model	RECALL		PRECISION		MRR		NDCG	
		L2	IP	L2	IP	L2	IP	L2	IP
1	Ada-002	52.01	51.98	52.01	51.98	52.01	51.98	52.01	51.98
	Jina-V2	41.48	4.66	41.48	4.66	41.48	4.66	41.48	4.66
	Jina-V3	54.69	54.67	54.69	54.67	54.69	54.67	54.69	54.67
	Nomic	42.67	43.13	42.67	43.13	42.67	43.13	42.67	43.13
	Qwen3-0.6B	**59.45**	**59.45**	**59.45**	**59.45**	**59.45**	**59.45**	**59.45**	**59.45**
5	Ada-002	62.71	62.66	12.54	12.53	56.17	56.08	57.81	57.73
	Jina-V2	51.85	12.12	10.37	2.42	45.47	7.31	47.07	8.50
	Jina-V3	66.09	66.22	13.22	13.24	59.09	59.11	60.84	60.89
	Nomic	54.69	55.11	10.94	11.02	47.30	47.75	49.15	49.59
	Qwen3-0.6B	**70.55**	**71.23**	**14.11**	**14.25**	**63.77**	**64.02**	**65.47**	**65.82**
10	Ada-002	65.91	65.94	6.59	6.59	56.61	56.53	58.86	58.80
	Jina-V2	55.58	15.43	5.56	1.54	45.97	7.76	48.28	9.58
	Jina-V3	69.94	70.13	6.99	7.01	59.61	59.64	62.10	62.16
	Nomic	58.92	59.24	5.89	5.92	47.87	48.30	50.53	50.93
	Qwen3-0.6B	**74.00**	**74.58**	**7.40**	**7.46**	**64.24**	**64.47**	**66.59**	**66.91**
25	Ada-002	69.74	69.93	2.79	2.80	56.86	56.79	59.80	59.78
	Jina-V2	59.63	21.37	2.39	0.85	46.23	8.12	49.27	11.02
	Jina-V3	73.92	74.08	2.96	2.96	59.87	59.89	63.08	63.13
	Nomic	64.41	64.70	2.58	2.59	48.22	48.64	51.87	52.26
	Qwen3-0.6B	**77.79**	**78.53**	**3.11**	**3.14**	**64.48**	**64.73**	**67.52**	**67.89**

7.3 Experiment 2: Reranking Strategies

This experiment evaluated the benefit of adding a reranking step after the initial retrieval. A reranker, typically a cross-encoder model, re-evaluates the top-k documents returned by the initial retriever, providing a more accurate relevance score. This allows for the rescue of relevant documents that may have been ranked lower by the initial semantic search.

7.3.1 Reranking Models Compared

We compared three cross-encoder models to rerank the top 500 candidates returned by the similarity search:

- **No Reranker (Baseline):** The initial retrieval order is used.

- **GTE ML Reranker Base:** A powerful reranker based on the DeBERTa-v3 architecture, known for its strong performance on various reranking tasks [21].
- **Jina Reranker v1 Tiny EN:** A lightweight and efficient reranker from Jina AI, designed for speed and scalability [22].
- **MXBAI Rerank Base v2:** A multilingual reranker that has shown competitive performance on diverse datasets [23].

7.3.2 Results and Discussion

The results in Table 7.6 and the visualizations in Figure 7.1 reveal a complex interplay between reranking models, chunking strategies, and thresholding methods. The key takeaway is that the effectiveness of a reranker is not absolute but is highly dependent on the subsequent thresholding strategy used to select the final documents.

When no intelligent thresholding is applied (Baseline) or when suboptimal methods like GMM or Otsu are used, the performance gains from reranking are minimal or even negative. These methods often pass too many documents (as seen in Table 7.7), diluting the context with noise and negating the precision benefits of the reranker. For instance, with the Baseline threshold, the F1 scores remain low (0.021-0.062) across all models because the high recall is undermined by extremely low precision.

However, the true power of reranking is unlocked when combined with an effective thresholding method like **Max Gap**. With this method, the **GTE ML Reranker Base** model achieves the highest F1 score (**0.654** for Full Page, **0.646** for Page Split), demonstrating a dramatic improvement over the No Reranker baseline (0.456 and 0.598). This is because the Max Gap method is highly effective at identifying the point of diminishing returns in the reranked list, selecting a small, highly relevant set of documents. As shown in Table 7.7, the GTE reranker with Max Gap passes an average of only 4.76 items, leading to a high precision of 0.594.

Conversely, the **MXBAI Rerank Base v2** model shows poor performance with aggressive thresholding methods like Knee and Max Gap. This is because the model's score distribution is very flat, making it difficult for these methods to find a clear cutoff point. As a result, it passes too few documents (e.g., only 4.38 items with Knee), leading to a collapse in recall (0.388) and a low F1 score (0.105).

In conclusion, this experiment demonstrates that simply adding a reranker is not enough. The combination of a powerful reranker like **GTE ML Reranker Base**

Table 7.6: Comparison of Reranker Performance Across Different Chunking Strategies and Thresholding Methods. The highest score in each metric column is highlighted in bold.

		Full Page			Page Split		
Threshold	Model	F1 Score	Precision	Recall	F1 Score	Precision	Recall
Baseline	No Reranker	0.021	0.011	0.967	0.021	0.011	0.918
	GTE ML Reranker Base	0.024	0.012	0.871	0.027	0.014	0.840
	Jina Reranker v1 Tiny EN	0.023	0.012	0.919	0.021	0.011	0.891
	MXBAI Rerank Base v2	0.062	0.033	0.947	0.060	0.032	0.880
GMM	No Reranker	0.016	0.008	0.986	0.014	0.007	0.920
	GTE ML Reranker Base	0.008	0.004	**0.988**	0.006	0.003	0.924
	Jina Reranker v1 Tiny EN	0.023	0.012	0.976	0.020	0.010	**0.924**
	MXBAI Rerank Base v2	0.009	0.005	0.986	0.008	0.004	0.920
Knee	No Reranker	0.082	0.044	0.881	0.081	0.043	0.878
	GTE ML Reranker Base	0.108	0.059	0.936	0.095	0.052	0.871
	Jina Reranker v1 Tiny EN	0.107	0.060	0.838	0.090	0.049	0.862
	MXBAI Rerank Base v2	0.105	0.066	0.388	0.092	0.059	0.347
Max Gap	No Reranker	0.456	0.404	0.681	0.598	0.548	0.751
	GTE ML Reranker Base	**0.654**	**0.594**	0.867	**0.646**	**0.601**	0.797
	Jina Reranker v1 Tiny EN	0.435	0.379	0.893	0.468	0.410	0.800
	MXBAI Rerank Base v2	0.071	0.052	0.986	0.055	0.037	0.918
Otsu	No Reranker	0.021	0.011	0.986	0.017	0.008	0.918
	GTE ML Reranker Base	0.018	0.009	**0.988**	0.009	0.005	0.922
	Jina Reranker v1 Tiny EN	0.035	0.019	0.979	0.033	0.018	0.920
	MXBAI Rerank Base v2	0.012	0.006	0.986	0.011	0.006	0.922
Percentile	No Reranker	0.037	0.019	0.952	0.036	0.018	0.907
	GTE ML Reranker Base	0.038	0.019	0.974	0.035	0.018	0.900
	Jina Reranker v1 Tiny EN	0.038	0.019	0.945	0.035	0.018	0.902
	MXBAI Rerank Base v2	0.037	0.019	0.971	0.035	0.018	0.902
2nd Derivative	No Reranker	0.254	0.199	0.721	0.262	0.198	0.776
	GTE ML Reranker Base	0.305	0.237	0.845	0.266	0.203	0.806
	Jina Reranker v1 Tiny EN	0.222	0.177	0.881	0.235	0.184	0.820
	MXBAI Rerank Base v2	0.059	0.042	0.981	0.049	0.035	0.916

with an aggressive, well-suited thresholding method like **Max Gap** is what yields the best performance, maximizing precision without sacrificing essential recall.

7.4 Experiment 3: Prompt Engineering to Prevent Hallucination

This experiment focuses on the generation part of the RAG pipeline, evaluating a wide range of Large Language Models to serve as the generator. The primary goal was to find the best-performing model and the optimal prompt and temperature configuration for our specific use case.

7.4.1 Motivation and Baseline

The evaluation started with a baseline configuration of **GPT-4o** using prompt `P1` and a temperature of `0.2`. As new and more powerful models were released during the course of this research, they were systematically evaluated against this baseline. This continuous evaluation process allowed us to stay at the cutting edge and select the most suitable model for integration into a production RAG system, ensuring that any replacement would offer superior performance.

7.4.2 Evaluation Methodology

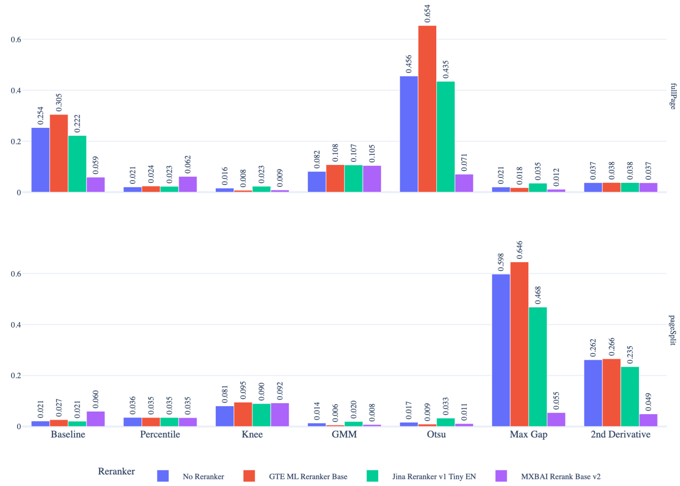
We used the **DeepEval** framework [24], which employs an LLM-as-a-Judge approach to score the generator's output. For subjective, use-case-specific evaluations, we utilized DeepEval's **G-Eval** functionality, which is inspired by the G-Eval framework [25] and uses a chain-of-thought process to evaluate outputs against custom criteria. We defined two such metrics:

- **Correctness (C):** This metric determines whether the actual output is factually correct based on the expected output. It heavily penalizes the omission of key details and the inclusion of information that contradicts the expected output.

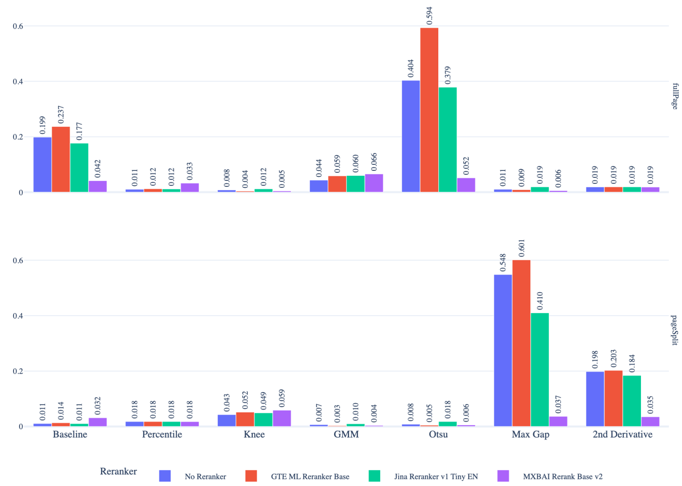
- **Specific Information Accuracy (SIA):** This metric evaluates whether the model's response appropriately uses information from the context without introducing specific details (names, places, numbers) that are not explicitly provided. It is particularly important for testing the model's ability to recognize when a question cannot be answered from the given context. For example, a high SIA score is awarded if the model correctly states it cannot answer a

Table 7.7: Comparison of the average number of items and unique documents passed by each thresholding method, faceted by chunking strategy and reranker model. Lower is better. The lowest value in each column is highlighted in bold.

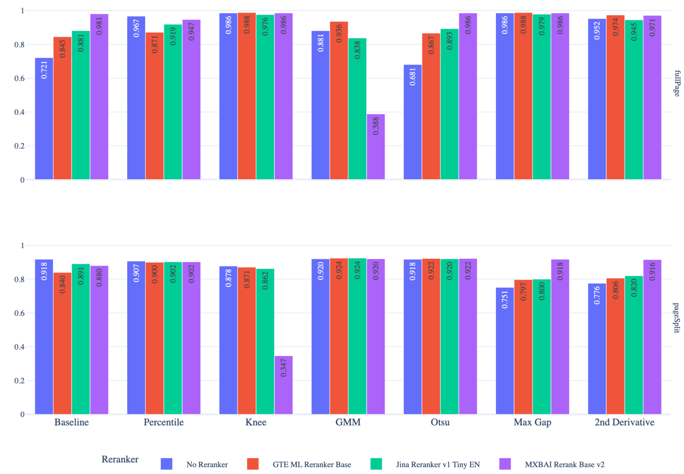
	Chunking	Full Page		Page Split	
		Docs Passed	Items Passed	Docs Passed	Items Passed
Threshold	Model				
Baseline	No Reranker	7.13	104.00	5.43	101.14
	GTE ML Reranker Base	6.28	79.70	5.46	68.79
	Jina Reranker v1 Tiny EN	7.13	94.76	5.91	103.57
	MXBAI Rerank Base v2	6.40	102.88	5.42	109.06
GMM	No Reranker	7.82	158.96	6.54	161.30
	GTE ML Reranker Base	11.97	287.73	10.73	340.32
	Jina Reranker v1 Tiny EN	7.82	164.49	6.71	159.13
	MXBAI Rerank Base v2	10.02	242.08	8.65	247.35
Knee	No Reranker	3.17	26.28	2.70	25.82
	GTE ML Reranker Base	3.58	21.39	3.23	22.02
	Jina Reranker v1 Tiny EN	3.26	20.05	3.16	25.56
	MXBAI Rerank Base v2	**1.06**	**4.38**	**1.11**	4.50
Max Gap	No Reranker	1.39	4.68	1.12	**2.36**
	GTE ML Reranker Base	1.40	4.76	1.22	11.87
	Jina Reranker v1 Tiny EN	2.99	93.73	1.69	34.14
	MXBAI Rerank Base v2	8.10	213.62	6.49	185.26
Otsu	No Reranker	6.16	126.08	5.20	128.48
	GTE ML Reranker Base	8.70	175.23	8.99	232.65
	Jina Reranker v1 Tiny EN	6.75	139.35	5.56	137.89
	MXBAI Rerank Base v2	9.86	231.67	7.88	210.58
Percentile	No Reranker	4.33	50.02	3.53	50.00
	GTE ML Reranker Base	5.12	50.12	4.80	50.12
	Jina Reranker v1 Tiny EN	5.24	49.71	4.30	50.01
	MXBAI Rerank Base v2	4.55	51.59	4.12	51.31
2nd Derivative	No Reranker	1.78	12.42	1.77	12.61
	GTE ML Reranker Base	2.46	27.72	2.61	53.69
	Jina Reranker v1 Tiny EN	4.49	161.05	3.09	99.01
	MXBAI Rerank Base v2	8.45	217.15	6.79	188.22



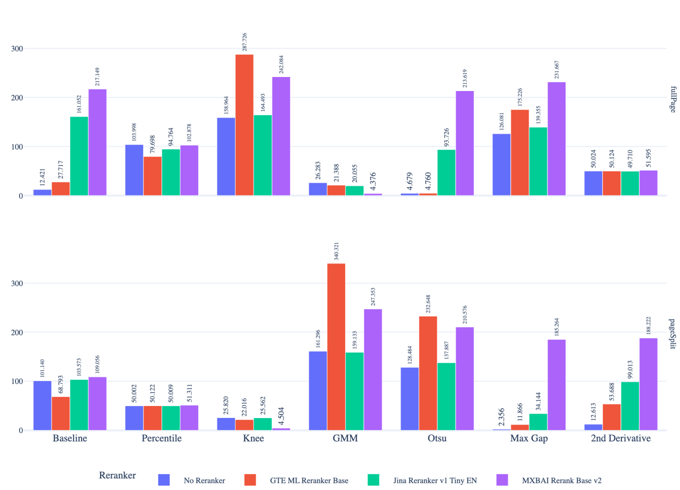
(a) Average F1 score by facet



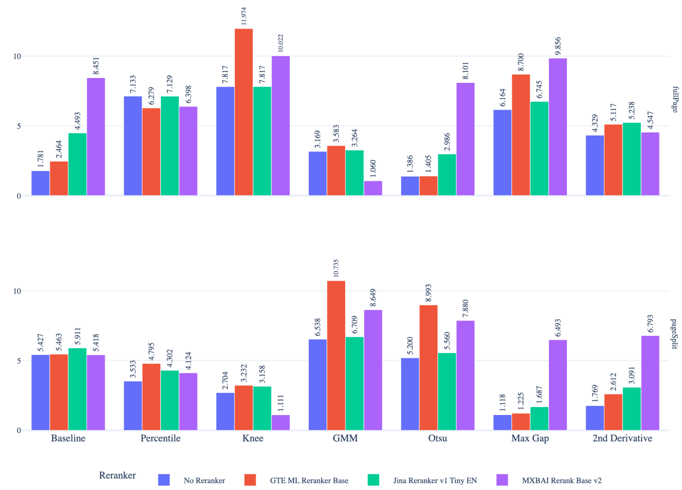
(b) Average precision by facet



(c) Average recall by facet



(d) Average number of items passed by facet



(e) Average number of retrieved documents passed by facet

Figure 7.1: Comparison of average metrics by model facet: (a) F1 score, (b) precision, (c) recall, (d) number of items passed, and (e) number of retrieved documents passed.

question like "Who is Cinderella?" when given an anonymized version of the story, as this demonstrates it is not relying on its own parametric knowledge.

In addition to our custom G-Eval metrics, we used several of DeepEval's standard metrics to assess other critical qualities of the generated output:

- **Answer Relevancy (AR):** This metric measures how relevant the generated output is to the user's input query. It ensures that the model's response is on-topic and directly addresses the question asked [24].
- **Faithfulness (F):** This metric evaluates whether the generated output factually aligns with the information present in the retrieved context. A high score indicates that the model did not invent facts and stayed true to the source material [24].
- **Hallucination (H):** This metric determines if the model generates factually incorrect information by comparing the output to the provided context. It is a crucial measure for ensuring the trustworthiness of the RAG system [24].

7.4.3 Results and Discussion

The comprehensive results are presented in Table 7.8. The baseline configuration (GPT-4o, P1, Temp 0.2) achieved a total score of 370.52. The results show that several newer models and prompt configurations were able to significantly outperform this baseline.

For instance, the **O4 Mini** model using prompt `P2` and a temperature of `0.2` achieved the highest total score of **403.86**. This demonstrates the value of continuous evaluation, as it allowed us to identify a model that not only performs better than the original baseline but also to determine the ideal prompt (`P2`) and temperature (`0.2`) for it. These findings are critical for deploying a RAG system that is not only accurate and faithful but also robust in handling queries that cannot be answered from the available context.

Table 7.8: DeepEval Generative Model and Prompt Evaluation Results

Model	Prompt	Temp	Avg. Scores					Total	GPT-4o					Claude 3.5				
			AR	C	F	H	SIA		AR	C	F	H	SIA	AR	C	F	H	SIA
Claude 3 Haiku	P1	0.0	57.70	66.66	87.18	69.23	74.36	355.13	61.54	69.23	84.62	64.10	79.49	53.85	64.10	89.74	74.36	69.23
Claude 3 Haiku	P1	0.2	62.82	61.54	87.18	65.38	67.94	344.86	61.54	53.85	84.62	58.97	64.10	64.10	69.23	89.74	71.79	71.79
Claude 3 Haiku	P2	0.0	66.66	62.82	89.74	67.94	74.36	361.52	64.10	56.41	89.74	64.10	82.05	69.23	69.23	89.74	71.79	66.67
Claude 3 Haiku	P2	0.2	62.82	61.53	79.49	65.38	73.07	342.29	61.54	58.97	79.49	58.97	82.05	64.10	64.10	79.49	71.79	64.10
Claude 3.5 Sonnet	P1	0.0	58.97	76.93	92.31	74.35	71.79	374.35	61.54	69.23	97.44	58.97	71.79	56.41	84.62	87.18	89.74	71.79
Claude 3.5 Sonnet v2	P1	0.0	48.72	75.65	96.16	66.66	67.95	355.14	48.72	66.67	92.31	51.28	69.23	48.72	84.62	100.00	82.05	66.67
Claude 3.5 Sonnet v2	P1	0.2	52.56	69.23	96.16	71.80	69.23	358.98	53.85	53.85	94.87	56.41	76.92	51.28	84.62	97.44	87.18	61.54
Claude 3.5 Sonnet v2	P2	0.0	62.82	70.52	91.03	78.20	84.62	387.19	64.10	61.54	89.74	61.54	84.62	61.54	79.49	92.31	94.87	84.62
Claude 3.5 Sonnet v2	P2	0.2	64.10	75.64	94.87	76.92	88.46	399.99	71.79	74.36	94.87	61.54	89.74	56.41	76.92	94.87	92.31	87.18
Claude 3.7 Sonnet	P1	0.0	48.72	82.05	88.47	71.80	78.20	369.24	48.72	69.23	92.31	56.41	74.36	48.72	94.87	84.62	87.18	82.05
Claude 3.7 Sonnet	P1	0.2	44.87	88.46	88.46	73.08	88.46	383.33	46.15	87.18	94.87	58.97	94.87	43.59	89.74	82.05	87.18	82.05
Claude 3.7 Sonnet	P2	0.0	48.72	78.21	93.59	78.20	85.90	384.62	53.85	71.79	94.87	61.54	79.49	43.59	84.62	92.31	94.87	92.31
Claude 3.7 Sonnet	P2	0.2	43.59	84.62	89.74	79.49	92.31	389.75	46.15	82.05	89.74	66.67	92.31	41.03	87.18	89.74	92.31	92.31
Claude 4.0 Sonnet	P1	0.2	38.46	83.34	89.75	76.92	83.33	371.80	38.46	79.49	87.18	61.54	89.74	38.46	87.18	92.31	92.31	76.92
Claude 4.0 Sonnet	P2	0.2	42.31	84.62	91.03	79.49	91.03	388.48	41.03	84.62	97.44	66.67	94.87	43.59	84.62	84.62	92.31	87.18
GPT-4 Omni	P1	0.0	56.41	65.38	93.59	70.52	67.94	353.84	53.85	58.97	92.31	56.41	71.79	58.97	71.79	94.87	84.62	64.10
GPT-4 Omni	P1	0.2	60.26	78.20	92.31	67.95	71.80	370.52	53.85	74.36	94.87	51.28	69.23	66.67	82.05	89.74	84.62	74.36
GPT-4 Omni	P2	0.0	56.41	74.36	92.31	61.54	78.20	362.82	51.28	64.10	92.31	48.72	76.92	61.54	84.62	92.31	74.36	79.49
GPT-4 Omni	P2	0.2	58.97	73.08	92.31	67.95	79.48	371.79	53.85	61.54	92.31	56.41	82.05	64.10	84.62	92.31	79.49	76.92
GPT-4 Omni Mini	P1	0.0	61.54	73.08	91.03	70.52	66.66	362.83	51.28	71.79	89.74	56.41	69.23	71.79	74.36	92.31	84.62	64.10
GPT-4 Omni Mini	P1	0.2	61.54	71.80	94.88	69.23	70.52	367.97	56.41	69.23	92.31	53.85	74.36	66.67	74.36	97.44	84.62	66.67
GPT-4 Omni Mini	P2	0.0	65.38	74.36	92.31	57.69	75.64	365.38	66.67	69.23	92.31	43.59	79.49	64.10	79.49	92.31	71.79	71.79
GPT-4 Omni Mini	P2	0.2	64.10	76.92	91.03	58.97	80.77	371.79	64.10	71.79	87.18	46.15	82.05	64.10	82.05	94.87	71.79	79.49
GPT-4.1	P1	0.2	65.38	82.05	93.59	73.07	85.89	399.98	64.10	82.05	97.44	56.41	89.74	66.67	82.05	89.74	89.74	82.05

AR: Answer Relevancy, **C**: Correctness, **F**: Faithfulness, **H**: Hallucination, **SIA**: Specific Information Accuracy

Continued on next page

Table 7.8 – continued from previous page

Model	Prompt	Temp	AR	C	F	H	SIA	Total	AR	C	F	H	SIA	AR	C	F	H	SIA
GPT-4.1	P2	0.2	62.82	80.77	93.59	70.52	89.74	397.44	61.54	76.92	94.87	61.54	89.74	64.10	84.62	92.31	79.49	89.74
GPT-4.1 Nano	P1	0.2	56.41	69.23	93.59	75.64	73.08	367.95	58.97	61.54	92.31	58.97	71.79	53.85	76.92	94.87	92.31	74.36
GPT-4.1 Nano	P2	0.2	65.38	57.69	87.18	69.23	78.20	357.68	64.10	51.28	89.74	58.97	79.49	66.67	64.10	84.62	79.49	76.92
Gemini 1.5 Flash	P1	0.0	73.08	61.53	93.59	62.82	65.39	356.41	69.23	58.97	92.31	51.28	61.54	76.92	64.10	94.87	74.36	69.23
Gemini 1.5 Flash	P1	0.2	73.08	57.69	96.16	61.54	67.95	356.42	71.79	51.28	94.87	48.72	69.23	74.36	64.10	97.44	74.36	66.67
Gemini 1.5 Flash	P2	0.0	78.20	52.56	96.16	69.23	67.95	364.10	76.92	51.28	97.44	53.85	66.67	79.49	53.85	94.87	84.62	69.23
Gemini 1.5 Flash	P2	0.2	78.20	67.95	92.31	69.23	82.06	389.75	82.05	66.67	92.31	53.85	79.49	74.36	69.23	92.31	84.62	84.62
Gemini 1.5 Pro	P1	0.0	74.36	60.26	93.59	65.39	56.41	350.01	74.36	53.85	100.00	43.59	53.85	74.36	66.67	87.18	87.18	58.97
Gemini 1.5 Pro	P1	0.2	75.64	52.56	91.03	64.11	55.13	338.47	74.36	48.72	94.87	43.59	56.41	76.92	56.41	87.18	84.62	53.85
Gemini 1.5 Pro	P2	0.0	67.94	56.41	97.44	65.38	65.38	352.55	64.10	51.28	97.44	51.28	71.79	71.79	61.54	97.44	79.49	58.97
Gemini 1.5 Pro	P2	0.2	74.36	58.97	93.59	60.25	74.36	361.53	71.79	58.97	92.31	46.15	74.36	76.92	58.97	94.87	74.36	74.36
Gemini 2.0 Flash	P1	0.0	48.72	48.72	97.44	65.39	53.84	314.11	51.28	46.15	97.44	43.59	48.72	46.15	51.28	97.44	87.18	58.97
Gemini 2.0 Flash	P1	0.2	47.44	46.16	100.00	58.98	56.41	308.99	46.15	41.03	100.00	33.33	48.72	48.72	51.28	100.00	84.62	64.10
Gemini 2.0 Flash	P2	0.0	66.66	61.53	93.59	67.95	74.36	364.09	64.10	58.97	92.31	56.41	71.79	69.23	64.10	94.87	79.49	76.92
Gemini 2.0 Flash	P2	0.2	73.08	56.41	92.31	65.38	75.64	362.82	71.79	51.28	92.31	51.28	74.36	74.36	61.54	92.31	79.49	76.92
Gemini 2.5 Flash	P2	0.2	55.12	74.36	91.03	69.23	85.90	375.64	51.28	66.67	92.31	56.41	87.18	58.97	82.05	89.74	82.05	84.62
Gemini 2.5 Flash	P2	0.2	57.70	75.64	91.03	74.35	87.18	385.90	53.85	74.36	94.87	58.97	89.74	61.54	76.92	87.18	89.74	84.62
Gemini 2.5 Flash	P2	0.2	62.82	78.20	91.03	70.52	87.18	389.75	58.97	76.92	92.31	56.41	89.74	66.67	79.49	89.74	84.62	84.62
Gemini 2.5 Flash	P2	0.2	66.66	74.36	89.75	69.23	88.46	388.46	58.97	66.67	92.31	53.85	87.18	74.36	82.05	87.18	84.62	89.74
Gemini 2.5 Pro	P2	0.2	62.82	82.05	93.59	73.08	91.03	402.57	61.54	74.36	94.87	61.54	94.87	64.10	89.74	92.31	84.62	87.18
Gemini 2.5 Pro	P2	0.2	62.82	82.05	89.74	71.80	88.46	394.87	66.67	71.79	89.74	58.97	89.74	58.97	92.31	89.74	84.62	87.18
Gemini 2.5 Pro	P2	0.2	57.69	76.93	92.31	75.64	87.18	389.75	58.97	69.23	97.44	61.54	87.18	56.41	84.62	87.18	89.74	87.18
Gemini 2.5 Pro	P2	0.2	62.82	78.21	88.47	76.92	87.18	393.60	58.97	69.23	92.31	64.10	89.74	66.67	87.18	84.62	89.74	84.62
O1	P1	0.0	66.67	82.05	93.59	76.92	84.62	403.85	71.79	76.92	94.87	56.41	87.18	61.54	87.18	92.31	97.44	82.05
O1	P2	0.0	70.52	71.79	97.44	62.82	93.59	396.16	66.67	64.10	97.44	46.15	92.31	74.36	79.49	97.44	79.49	94.87
O3	P1	0.2	75.64	66.67	88.47	69.23	78.20	378.21	76.92	74.36	61.54	71.79	84.62	92.31	53.85	84.62	76.92	79.49

AR: Answer Relevancy, **C**: Correctness, **F**: Faithfulness, **H**: Hallucination, **SIA**: Specific Information Accuracy

Continued on next page

Table 7.8 – continued from previous page

Model	Prompt	Temp	AR	C	F	H	SIA	Total	AR	C	F	H	SIA	AR	C	F	H	SIA
O3	P2	0.2	70.51	80.77	91.03	73.07	85.89	401.27	71.79	71.79	89.74	56.41	82.05	69.23	89.74	92.31	89.74	89.74
O3 Mini	P1	0.0	70.52	67.95	96.16	74.36	62.82	371.81	74.36	56.41	94.87	61.54	66.67	66.67	79.49	97.44	87.18	58.97
O3 Mini	P1	0.2	60.25	65.38	93.59	67.95	65.39	352.56	64.10	48.72	94.87	51.28	61.54	56.41	82.05	92.31	84.62	69.23
O3 Mini	P2	0.0	64.11	74.36	92.31	74.36	75.64	380.78	61.54	69.23	89.74	66.67	74.36	66.67	79.49	94.87	82.05	76.92
O3 Mini	P2	0.2	65.38	71.80	96.16	69.23	79.49	382.06	66.67	61.54	94.87	56.41	79.49	64.10	82.05	97.44	82.05	79.49
O4 Mini	P1	0.2	67.95	75.64	93.59	73.07	83.34	393.59	69.23	66.67	76.92	74.36	94.87	92.31	56.41	89.74	84.62	82.05
O4 Mini	P2	0.2	74.36	83.34	93.59	66.67	85.90	403.86	71.79	79.49	92.31	48.72	87.18	76.92	87.18	94.87	84.62	84.62

AR: Answer Relevancy, **C**: Correctness, **F**: Faithfulness, **H**: Hallucination, **SIA**: Specific Information Accuracy

7.4.4 Comparative Analysis Against GPT-4o

To better understand the relative performance of each model, we calculated the delta between each model's total score and the baseline score of GPT-4o (370.52). The results are summarized in Table 7.9.

Table 7.9: Total Score Delta vs. GPT-4o (Highest Scores)

Model	**Prompt**	**Temp**	**Delta from GPT-4o**
O4 Mini	P2	0.2	+33.34
O1	P1	0.0	+33.33
Gemini 2.5 Pro	P2	0.2	+32.05
O3	P2	0.2	+30.75
Claude 3.5 Sonnet v2	P2	0.2	+29.47
GPT-4.1	P1	0.2	+29.46
Claude 3.7 Sonnet	P2	0.2	+19.23
Gemini 1.5 Flash	P2	0.2	+19.23
Gemini 2.5 Flash	P2	0.2	+19.23
Claude 4.0 Sonnet	P2	0.2	+17.96
O3 Mini	P2	0.2	+11.54
Claude 3.5 Sonnet	P1	0.0	+3.83
GPT-4 Omni Mini	P2	0.2	+1.27
GPT-4 Omni	P2	0.2	+1.27
GPT-4o Baseline Total Score: 370.52			
GPT-4.1 Nano	P1	0.2	-2.57
Gemini 2.0 Flash	P2	0.0	-6.43
Claude 3 Haiku	P2	0.0	-9.00
Gemini 1.5 Pro	P2	0.2	-9.00

Inference from the Results

The delta analysis reveals several key insights. Firstly, the top-performing models, such as **O4 Mini**, **O1**, and **Gemini 2.5 Pro**, show a significant improvement over the GPT-4o baseline, with score increases exceeding 30 points. This underscores the rapid advancements in generative models, where newer architectures can yield substantial performance gains.

Secondly, the choice of prompt and temperature settings is evidently crucial. For many models, there is a considerable performance variance between different prompt versions (P1 vs. P2) and temperature settings. For instance, the O1 model with prompt P1 scores much higher than with P2 (403.85 vs 396.16: +7.69 points). This highlights the necessity of meticulous prompt engineering and parameter tuning to unlock a model's full potential.

Chapter 8

Conclusions and Future Work

This thesis has provided a comprehensive study of Retrieval-Augmented Generation methods, systematically exploring the components and strategies that contribute to building more robust and reliable Large Language Models. Our work has demonstrated that RAG is a powerful paradigm for mitigating the inherent weaknesses of LLMs, namely knowledge cutoff and hallucination, by grounding them in external, verifiable information.

Our investigation began with the fundamentals of the RAG pipeline, establishing a clear understanding of the interplay between the retrieval and generation stages. We then delved into the critical optimization techniques at each step. The experimental results clearly indicate that the performance of a RAG system is not determined by a single component, but by the careful tuning and synergistic combination of multiple factors.

Our key findings can be summarized as follows:

- **Systematic Optimization is Key:** The dramatic performance improvement between our initial baseline and the final optimized system underscores the necessity of a component-wise approach to building RAG pipelines. Naive or default configurations, as defined by Gao et al. (2024) [2], are unlikely to yield optimal results. Our experiments showed that moving from a baseline ‘ada-002‘ embedding model to ‘Qwen3-Embedding-0.6B‘ and adding a ‘GTE ML Reranker Base‘ with ‘Max Gap‘ thresholding improved the F1 score from 0.021 to 0.654.

- **Retrieval Quality is Paramount:** The experiments consistently showed that improvements in the retrieval stage—through better embedding models and powerful reranking—had a significant downstream impact on the quality of the final generated answer. The top-performing embedding model, ‘Qwen3-Embedding-0.6B‘, outperformed the baseline ‘ada-002‘ by a large margin in all metrics.

- **Reranking Offers Significant Gains:** The addition of a reranking step, particularly with a sophisticated cross-encoder model like ‘GTE ML Reranker Base‘, was shown to be one of the most effective ways to boost retrieval precision. This confirms the value of post-retrieval processing in the Advanced RAG paradigm.

- **Generator Choice and Prompting are Crucial:** The extensive evaluation of generative models showed that the choice of the LLM and the prompt style has a profound effect on the final output. Newer models like ‘O4 Mini‘ and ‘Gemini 2.5 Pro‘, when paired with the right prompt and temperature settings, can significantly outperform older baselines like GPT-4o in terms of faithfulness, relevancy, and adherence to instructions, with score increases of over 30 points in our evaluation framework.

In essence, this work has shown that building a state-of-the-art RAG system is a multi-faceted engineering challenge that requires careful consideration of the trade-offs between performance, cost, and complexity at each stage of the pipeline.

8.1 Future Work

The field of Retrieval-Augmented Generation is rapidly evolving, and this study opens up several avenues for future research, many of which align with the directions proposed by Gao et al. (2024) [2]:

- **Adaptive RAG Architectures:** Future systems could dynamically adjust their strategies based on the query, as suggested by the Modular RAG paradigm [2]. For example, a simple query might only require a basic retrieval step, while a complex, multi-faceted query could trigger a more sophisticated pipeline involving multiple retrievers and a cross-encoder reranker. This would optimize for both efficiency and quality.

- **Advanced Chunking and Indexing:** Exploring more advanced, model-based chunking strategies and the use of multi-vector indexing (where chunks are represented by multiple vectors capturing different aspects of their meaning) could lead to further improvements in retrieval relevance.

- **Graph-Based RAG:** Integrating knowledge graphs as the retrieval backbone could provide more structured and reliable information, especially in well-defined domains. Future work could explore hybrid systems that combine the strengths of both vector-based and graph-based retrieval.

- **Fine-tuning and Self-Correction:** Developing tighter feedback loops where the generator's output is used to fine-tune the retriever and the embedding models could lead to self-improving RAG systems. This could involve reinforcement learning techniques to reward the retriever for finding chunks that lead to high-quality answers.

- **Energy Efficiency and Sustainability:** As RAG systems become more widespread, investigating the energy consumption of different pipeline configurations will be an important area of research. Finding ways to build efficient yet powerful RAG systems is a key challenge for the future.

By pursuing these research directions, the community can continue to advance the capabilities of Retrieval-Augmented Generation, paving the way for even more powerful, reliable, and trustworthy AI systems.

Bibliography

[1] Patrick Lewis et al. "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks". In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle et al. Vol. 33. 2020, pp. 9459–9474.

[2] Yunfan Gao et al. *Retrieval-Augmented Generation for Large Language Models: A Survey*. 2024. arXiv: `2312.10997 [cs.CL]`.

[3] IBM Corporation. *Retrieval Augmented Generation*. Accessed: 2025-07-10. 2024. URL: `https://www.ibm.com/architectures/patterns/genai-rag`.

[4] Nelson F. Liu et al. *Lost in the Middle: How Language Models Use Long Contexts*. 2023. arXiv: `2307.03172 [cs.CL]`.

[5] Yair Diskin, Elad Hazan, and Ofer Ram. "RAG-Fusion: a New Take on Retrieval-Augmented Generation". In: *arXiv preprint arXiv:2401.10221* (2024). URL: `https://arxiv.org/abs/2401.10221`.

[6] 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`.

[7] Justin Zobel, Alistair Moffat, and Kotagiri Ramamohanarao. "Inverted files versus signature files for text indexing". In: *ACM Trans. Database Syst.* 23.4 (Dec. 1998), pp. 453–490. ISSN: 0362-5915. DOI: `10.1145/296854.277632`. URL: `https://doi.org/10.1145/296854.277632`.

[8] Omar Khattab and Matei Zaharia. "ColBERT: Efficient and Effective Passage Search via Contextualized Late Interaction over BERT". In: *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 2020, pp. 39–48. URL: `https://doi.org/10.1145/3397271.3401075`.

[9] Michael Günther et al. *Late Chunking: Contextual Chunk Embeddings Using Long-Context Embedding Models*. 2025. arXiv: 2409.04701 [cs.CL]. URL: https://arxiv.org/abs/2409.04701.

[10] Saba Sturua et al. *jina-embeddings-v3: Multilingual Embeddings With Task LoRA*. 2024. arXiv: 2409.10173 [cs.CL]. URL: https://arxiv.org/abs/2409.10173.

[11] Michael Günther et al. *jina-embeddings-v4: Universal Embeddings for Multimodal Multilingual Retrieval*. 2025. arXiv: 2506.18902 [cs.AI]. URL: https://arxiv.org/abs/2506.18902.

[12] Lianmin Zheng et al. *Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena*. 2023. arXiv: 2306.05685 [cs.CL].

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

[14] Hugo Touvron et al. *Llama 2: Open Foundation and Fine-Tuned Chat Models*. 2023. arXiv: 2307.09288 [cs.CL].

[15] Gemini Team, Rohan Anil, and et al. *Gemini: A Family of Highly Capable Multimodal Models*. 2023. arXiv: 2312.11805 [cs.CL].

[16] Dawei Zhu et al. *LongEmbed: Extending Embedding Models for Long Context Retrieval*. 2024. arXiv: 2404.12096 [cs.CL]. URL: https://arxiv.org/abs/2404.12096.

[17] Jianguo Wang et al. "Milvus: A Purpose-Built Vector Data Management System". In: *Proceedings of the 2021 International Conference on Management of Data*. New York, NY, USA: Association for Computing Machinery, 2021, pp. 2614–2627. ISBN: 9781450383431. URL: https://doi.org/10.1145/3448016.3457550.

[18] MTEB Authors. *Massive Text Embedding Benchmark (MTEB) Leaderboard*. Accessed on August 1, 2025. 2025. URL: https://huggingface.co/spaces/mteb/leaderboard (visited on 08/01/2025).

[19] An Yang et al. *Qwen3 Technical Report*. 2025. arXiv: 2505.09388 [cs.CL]. URL: https://arxiv.org/abs/2505.09388.

[20] Yanzhao Zhang et al. *Qwen3 Embedding: Advancing Text Embedding and Reranking Through Foundation Models*. 2025. arXiv: 2506.05176 [cs.CL]. URL: https://arxiv.org/abs/2506.05176.

[21] Zhenghao Li et al. "What makes a good reranker for dense retrieval?" In: *arXiv preprint arXiv:2404.01088* (2024).

[22] Jina AI. *jina-reranker-v1-tiny-en*. `https://huggingface.co/jinaai/jina-reranker-v1-tiny-en`. 2023.

[23] Ramith Pradeep et al. "Reranking with a Small Language Model". In: *arXiv preprint arXiv:2401.06405* (2024).

[24] Confident AI. *DeepEval: An open-source evaluation framework for LLMs*. `https://github.com/confident-ai/deepeval`. Accessed: 2025-07-01. 2023.

[25] Yang Liu et al. *G-Eval: NLG Evaluation using GPT-4 with Better Human Alignment*. 2023. arXiv: `2303.16634 [cs.CL]`.