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

A mio papà Alessandro, a mia mamma Mara, a mio fratello Gabriele, alla mia Alice e a tutta la mia famiglia: grazie per l'amore, il sostegno e la forza che mi avete donato, siete la mia radice e la mia ispirazione.

Contents

Chapter 1

Introduction

The advent of Large Language Models (LLMs) represents a significant advancement in the field of artificial intelligence, exhibiting remarkable capabilities in understanding and generating human-like text. These models, trained on extensive and diverse datasets, serve as foundational components for numerous applications, ranging from sophisticated chatbots to advanced content creation tools. However, their capabilities are subject to certain inherent 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 refers to the propensity of these models to produce plausible but factually inaccurate or incoherent information, a consequence of their probabilistic nature and their training objective to predict the next word rather than to assert factual statements.

This thesis investigates Retrieval-Augmented Generation (RAG), a paradigm developed to directly mitigate these limitations [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 addresses the knowledge cutoff issue by enabling access to real-time information. The core principle involves integrating the generative capabilities of LLMs with the factual accuracy of information retrieval systems, thereby yielding more robust, reliable, and contextually pertinent responses [2].

The RAG paradigm has undergone substantial evolution 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 systematically examine the foundational Naive RAG architecture and advance towards more sophisticated techniques, primarily focusing on the Advanced RAG paradigm. While Modular RAG is introduced as a concept, its detailed examination falls outside the scope of this thesis, serving instead as a direction for future work. We will conduct a comprehensive analysis of the critical elements comprising 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 methodology of designing 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 the influence of each of these components on system performance, with the objective of establishing a clear framework for constructing 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 establishes the foundational understanding by elucidating the core mechanics of RAG, its principal challenges, and the function of its key components.

2.1 How it Works

The foundational RAG architecture, frequently termed **Naive RAG** [2], functions through two primary stages: retrieval and generation, as initially posited by Lewis et al. (2020) [1]. This paradigm was significantly influenced by earlier and parallel developments in dense retrieval and knowledge-augmented language models, such as Dense Passage Retrieval (DPR) by Karpukhin et al. (2020) [3] and REALM (Retrieval-Augmented Language Model) by Guu et al. (2020) [4], which laid crucial groundwork for effectively integrating external knowledge. The process commences upon a user's query submission. Rather than directly inputting the query to the LLM, the RAG pipeline first intercepts and processes it by interacting with an external knowledge base.

1. **Retrieval Stage:** The primary objective of this stage is to efficiently iden-

tify and extract the most pertinent information from an extensive external knowledge base to facilitate the answering of the user's query. The user's query is transformed into a vector representation, termed the *embedding*, utilizing a text embedding model. This query embedding is then used to search a pre-indexed collection of documents. The objective is to identify and retrieve text segments that exhibit semantic similarity to the query. This retrieval is typically executed utilizing a vector database, which is engineered for high-throughput similarity searches across extensive datasets of embeddings. The output of this stage comprises a set of pertinent text segments, frequently designated as the *context*.

2. **Generation Stage:** In this stage, the Large Language Model (LLM) functions as a sophisticated text synthesizer. The objective of this stage is to generate a coherent and factually accurate response by leveraging the retrieved information, a process frequently termed knowledge-grounded generation [5]. 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, pertinent information, the LLM is directed to produce a response that is not only contextually appropriate but also substantiated by the facts contained within the retrieved documents, thereby minimizing reliance on its internal, potentially outdated or erroneous, knowledge. This process substantially enhances the accuracy and factuality of the output.

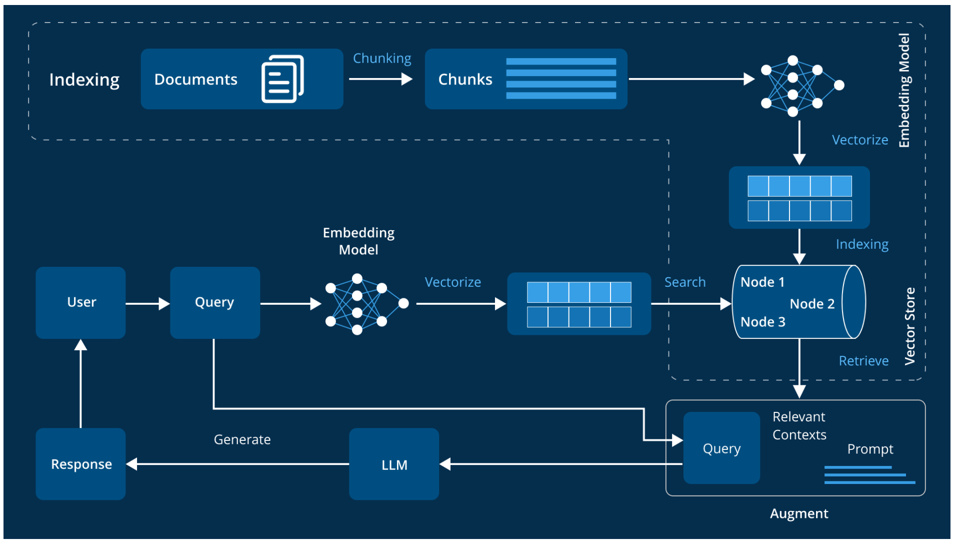


Figure 2.1: High-level architecture of a Retrieval-Augmented Generation system.

2.2 Main Challenges

The apparent simplicity of the Naive RAG pipeline conceals several intricate challenges that must be addressed to construct 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. Should the retriever yield irrelevant or suboptimal documents, the LLM may disregard them or, more critically, integrate erroneous information into its response. A significant challenge is the *lost in the middle problem*, wherein LLMs frequently fail to prioritize relevant information when it is interspersed with less pertinent segments within the context window [6]. This phenomenon highlights the imperative for retrieval systems that not only identify relevant documents but also effectively rank them.

2.2.2 Optimizing Retrieval vs. Generation Trade-offs

An intrinsic trade-off exists between the velocity and comprehensiveness of the retrieval step. Retrieving a greater number of documents may enhance the probability of identifying accurate information, yet it concurrently augments the computational burden and the potential for noise introduction. The permissible length of the context provided to the LLM is constrained by its inherent context window size. As emphasized by Gao et al. (2024) [2], optimizing the selection of the most pertinent document chunks from the initial similarity search constitutes a significant 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 frequently exhibits characteristics of disorder and heterogeneity. The retrieved context may encompass contradictory facts or extraneous details. The RAG system must demonstrate resilience to such noise, and the LLM must possess the capacity to synthesize information from multiple sources, identify contradictions, and prioritize the most reliable data. Advanced RAG architectures, such as those employing query transformations or reranking, are designed to tackle this issue [2, 7].

2.2.4 Seamless Integration and Synthesis

The LLM must be capable of seamlessly integrating the retrieved information into a coherent and linguistically natural response. This necessitates not merely the extraction of facts but also a comprehensive understanding of contextual nuances and their integration into a cohesive narrative that directly addresses the user's query. The fluency and relevance of the final output serve as direct indicators of the RAG system's efficacy.

2.3 Vector Databases and Similarity Search

The conceptualization of representing text as vectors within a multi-dimensional space for similarity comparison, foundational to modern vector databases, was initially advanced by the Vector Space Model (VSM) in 1975 [8]. Vector databases are a cornerstone of modern RAG systems, serving as the indexed knowledge base. These databases operate by storing textual data as high-dimensional vectors, commonly referred to as *embeddings*. Upon receipt of a query, it is similarly transformed into an embedding, and the database then identifies the vectors within its index that exhibit the highest proximity to the query vector.

2.3.1 Measuring Similarity

The predominant method for quantifying the distance between two vectors within the context of RAG is **cosine similarity**, which quantifies the cosine of the angle subtended by 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 enhanced efficiency, it is standard practice to **normalize** the vectors prior to their storage 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 undergo normalization, the denominator in the cosine similarity formula ($\|A\|\|B\|$) converges to 1. Consequently, the cosine similarity calculation simplifies to merely the **dot product** of the vectors, resulting in a substantially reduced computational cost:

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

This optimization obviates the computationally intensive square root operations required for calculating vector magnitudes, thereby enabling significantly accelerated similarity searches, a critical factor for real-time RAG applications.

2.3.3 Indexing for Efficient Search

To circumvent a brute-force search, vector databases employ specialized indexing algorithms for Approximate Nearest Neighbor (ANN) search. These algorithms, including Hierarchical Navigable Small World (HNSW) [9] and Inverted File (IVF) [10], facilitate the efficient retrieval of the top-k most similar vectors without necessitating a comparison of the query vector to every individual vector in the database. This is paramount for attaining low-latency responses in large-scale RAG systems.

Hierarchical Navigable Small World (HNSW)

HNSW graphs construct a multi-layer graph structure where each layer connects a subset of nodes from the layer below, with higher layers containing fewer, more globally connected nodes. Search queries start at the top layer, quickly navigating to a region of interest, and then descend to lower layers for finer-grained search. This hierarchical approach allows for very fast query times (low latency) and high recall, making it a popular choice for real-time applications. However, HNSW indices can be relatively large in terms of storage and have higher build costs compared to some other methods.

Inverted File (IVF)

IVF indexes, on the other hand, partition the vector space into Voronoi cells, with each cell represented by a centroid. During indexing, vectors are assigned to their nearest centroid. For a query, the search is narrowed down to a few relevant cells, and then a brute-force search is performed only within those cells. IVF offers a good balance between search speed and memory usage, and its index build time is generally faster than HNSW. However, its search performance can be more sensitive to the number of centroids and the distribution of data, and it typically offers lower recall than HNSW for a given search speed.

In terms of trade-offs, HNSW generally provides superior search performance (lower latency, higher recall) at the cost of higher memory consumption and potentially longer index build times. IVF offers a more memory-efficient solution with faster index construction, but may sacrifice some search accuracy or speed compared to HNSW, especially in very high-dimensional spaces [11]. For the experiments conducted in this thesis, HNSW will be utilized due to its excellent balance of speed and recall, which is critical for robust RAG system performance.

2.4 Mitigating Hallucinations

A primary advantage of RAG is its capacity to mitigate LLM hallucinations. By providing factual, verifiable context directly within the prompt, RAG anchors the model's response in empirical data. 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 does not present a complete panacea. Hallucinations within RAG systems can predominantly originate from two sources: retrieval failure (e.g., the provision of irrelevant or inaccurate context) and generation deficiency (e.g., the LLM's misinterpretation or disregard of the provided context). Should the retrieved context be of suboptimal quality, contain subtle inaccuracies, or prove inherently misleading, the LLM may nonetheless produce a deficient 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 constitutes the core of a RAG system. Its capacity to extract high-quality, pertinent information from an extensive corpus is the primary determinant of the system's overall performance. This chapter offers a comprehensive examination of the critical techniques employed to construct and optimize the retrieval pipeline, encompassing the initial processing of documents through 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 constitutes a crucial pre-retrieval processing step that entails segmenting large documents into smaller, more tractable units [12]. The objective is to generate chunks that are semantically coherent and sufficiently compact to be efficiently processed by embedding models and accommodated within the context window of an LLM. The selection of a chunking strategy exerts a substantial influence on retrieval quality.

3.1.1 Naive vs. Semantic Chunking

Naive Chunking, also referred to as fixed-size chunking, represents the most straightforward approach. It entails segmenting documents into portions of a predetermined length (e.g., 200 words) with an optional overlap between contiguous chunks. While straightforward to implement, this method can prove suboptimal as it frequently bisects sentences or paragraphs, thereby disrupting the semantic continuity of the text. This can lead to a loss of crucial context, as a single concept might be split across multiple chunks, making it harder for the embedding model to capture its full meaning and for the LLM to synthesize a coherent answer.

Semantic Chunking constitutes a more sophisticated approach. Rather than depending on arbitrary lengths, it endeavors to delineate the text at logical boundaries. This can be accomplished through several methodologies:

Sentence-Level Chunking

Employing natural language processing libraries to segment the text into discrete sentences. This approach offers high granularity, making it easier to pinpoint exact sources for generated responses. However, it can result in a large number of very small chunks, which might lack sufficient context on their own, potentially leading to less informative embeddings if a concept spans multiple sentences.

Recursive Chunking

A hierarchical method that initially attempts to segment by larger logical units (e.g., paragraphs, sections), subsequently by smaller units (e.g., sentences), and ultimately by words, with the aim of preserving semantic coherence to the greatest extent feasible. This method is particularly effective for documents with clear structural hierarchies, as it prioritizes maintaining the integrity of semantic units. For instance, a document might first be split by double newlines (paragraphs), then by single newlines, then by sentences, and finally by characters if necessary, ensuring that chunks are as semantically complete as possible.

The choice of chunking strategy directly impacts the quality of the generated embeddings and, consequently, the retrieval performance. Chunks that are too small may lack sufficient context for the embedding model to accurately capture their meaning, leading to poor retrieval. Conversely, chunks that are too large might dilute the relevance of specific information, making it harder for the retriever to identify the most pertinent segments and increasing the risk of the "lost in the middle" problem for the LLM. An optimal chunk size balances contextual completeness with conciseness.

From a practical standpoint, various libraries and tools offer functionalities for implementing these chunking strategies. For instance, frameworks like LangChain [13] provide robust text splitters that support fixed-size, recursive, and other semantic-aware chunking methods. Natural Language Toolkit (NLTK) [14] and

SpaCy [15] are commonly used for sentence tokenization, which forms the basis for sentence-level chunking.

3.2 Embedding Models

The selection of an embedding model is paramount for accurately capturing the semantic meaning of the text. These models convert text into high-dimensional vectors, such that semantically similar texts are positioned in closer proximity within the vector space.

3.2.1 Contextual Embeddings

Contemporary RAG systems leverage **contextual embeddings**, exemplified by those generated by transformer-based models including BERT [16], RoBERTa [17], and the OpenAI Ada series. In contrast to earlier static word embeddings (e.g., Word2Vec, GloVe), which attribute a singular vector to each word, contextual embeddings produce a distinct vector for a word contingent upon the sentence in which it is situated. This enables them to capture linguistic nuances, ambiguity, and the richness of language, thereby facilitating more accurate semantic search. The underlying transformer architecture, with its self-attention mechanisms, allows these models to weigh the importance of different words in a sentence when generating an embedding for a specific word, thus capturing its meaning in context. This capability is crucial for tasks requiring a deep understanding of semantic relationships, such as information retrieval.

3.2.2 Fine-tuning Embedding Models

For domain-specific applications, pre-trained embedding models may not exhibit optimal performance. Fine-tuning the embedding model on a dataset representative of the target domain can substantially enhance retrieval relevance [18]. This process customizes the model to the specific vocabulary and semantic relationships inherent in the corpus. As demonstrated by Chen et al. (2023) [18], this domain adaptation is particularly beneficial in specialized fields like legal, medical, or financial research, where the general-purpose embeddings might not fully capture the unique terminology and conceptual relationships. Fine-tuning typically involves training the pre-trained model on a new, smaller dataset that is highly relevant to the target domain. This can be achieved through various techniques, such as contrastive learning or triplet loss, where the model learns to pull semantically similar pairs closer together in the vector space while pushing dissimilar pairs further apart.

The quality of embedding models is often evaluated using benchmarks that assess their ability to capture semantic similarity and perform retrieval tasks. Metrics such as Mean Average Precision (MAP), Recall@k, and Normalized Discounted Cumulative Gain (nDCG) are commonly used to quantify their performance on various datasets. A prominent resource for this evaluation is the Massive Text Embedding Benchmark (MTEB) leaderboard [19]. MTEB comprehensively ranks over 100 text and image embedding models across more than 1000 languages, encompassing 131 tasks categorized into 9 task types (including BitextMining, Classification, Clustering, InstructionRetrieval, MultilabelClassification, PairClassification, Reranking, Retrieval, and STS). It also covers 20 diverse domains such as Academic, Financial, Legal, Medical, and News, providing a broad and highly-curated comparison of embedding model performance in various real-world scenarios.

3.3 Post-Retrieval Reranking and Filtering

Subsequent to the initial retrieval of documents based on semantic similarity, their relevance and ordering can be further refined through post-retrieval processing. This constitutes a pivotal component of the **Advanced RAG** paradigm [2].

3.3.1 BM25 and TF-IDF for Reranking

Traditional information retrieval algorithms, such as **BM25** [20] and **TF-IDF**, are predicated on keyword matching. They demonstrate high efficacy in identifying documents that contain the precise keywords from the query. While dense retrievers (vector search) ascertain the user's semantic intent, these sparse retrievers identify the user's explicit lexical terms. BM25, specifically, is a ranking function that estimates the relevance of documents to a given search query. It improves upon TF-IDF by incorporating document length normalization and term frequency saturation, which prevents very long documents or documents with very high term frequencies from being disproportionately ranked. By employing BM25 or TF-IDF to rerank the candidates retrieved via vector search, precision can be enhanced through the elevation of documents exhibiting substantial lexical overlap with the query [2]. This hybrid approach leverages the strengths of both semantic (dense) and keyword (sparse) matching.

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

A fully **hybrid system** integrates the scores derived from both dense (semantic) and sparse (keyword) retrieval methods from its inception [21]. A prevalent approach involves utilizing a weighted combination of scores from a vector search and a BM25 search to yield a final relevance score. This enables the system to capitalize on the strengths of both approaches, thereby encompassing both semantic relevance and keyword importance for a more robust retrieval process [2]. One common method for combining scores is Reciprocal Rank Fusion (RRF), which aggregates ranked lists from multiple retrieval methods without requiring score normalization, making it robust to different scoring scales. Hybrid systems are particularly effective in scenarios where queries might be ambiguous or where both semantic understanding and exact keyword matches are important for identifying relevant information.

3.3.3 Cross-Encoder Rerankers

For maximal accuracy, a **cross-encoder** model can be employed as a final reranking step [22]. In contrast to bi-encoders (standard embedding models) that generate distinct embeddings for the query and documents independently, a cross-encoder processes the query and a candidate document as a unified input. This allows the model to perform a deep, token-by-token comparison between the query and the document, yielding a highly precise relevance score [23]. This "cross-attention" mechanism enables the model to capture intricate relationships and subtle nuances that might be missed by bi-encoders. However, cross-encoders incur significant computational expense due to the need to process each query-document pair, making them generally reserved for reranking a limited number of top candidates (typically 50-200) from a more rapid, initial retrieval stage. Their use is justified in applications where high precision and relevance are paramount, even at the cost of increased latency.

3.4 Dynamic Similarity Thresholding

Rather than retrieving a predetermined number of chunks (top-N), **dynamic similarity thresholding** adjusts the retrieval process based on the query itself [24]. For certain queries, only a limited number of highly relevant chunks may be requisite, whereas for others, a more expansive context proves advantageous. Dynamic thresholding methods analyze the distribution of similarity scores for a given query and endeavor to identify a natural cutoff point, thereby facilitating the retrieval of a more contextually appropriate number of chunks. This precludes the inclusion of irrelevant documents when similarity scores exhibit a sharp decline and permits more comprehensive retrieval when numerous documents demonstrate comparable relevance.

3.4.1 Motivation for Dynamic Thresholding

The traditional approach of retrieving a fixed number of top-N documents (e.g., the top 5 most similar chunks) often proves suboptimal. A fixed N fails to account for the varying relevance distribution across different queries. For a highly specific query, only one or two documents might be truly relevant, and retrieving more would introduce noise. Conversely, for a broad query, a larger set of documents might be necessary to provide a comprehensive answer. Furthermore, a fixed N can exacerbate the "lost in the middle" problem, where relevant information might be overlooked by the LLM if it's buried within a large context window filled with less relevant chunks. Dynamic thresholding aims to retrieve precisely the right amount of information, optimizing for both precision and recall.

3.4.2 Methods for Dynamic Thresholding

Several approaches can be employed to implement dynamic similarity thresholding. In our experiments, we specifically evaluated the following methods:

Statistical Methods

These methods analyze the statistical properties of the similarity scores for a given query. For instance, one could set a threshold based on a certain number of standard deviations from the mean similarity score, or by identifying a significant drop-off in scores. A common example is **Percentile** thresholding, where a fixed percentile of the ranked scores is chosen as the cutoff. While straightforward, this method often included an excessive number of documents, as observed in our experiments (Table 7.7), leading to diluted context.

Distribution-based Methods

These approaches examine the distribution of similarity scores to find natural cut-off points. Techniques evaluated include:

- **Gaussian Mixture Models (GMM):** This method attempts to model the distribution of similarity scores as a mixture of Gaussian distributions, typically two (one for relevant, one for irrelevant), and sets the threshold at the point where the probability of belonging to the "relevant" distribution drops significantly. Our experiments showed GMM often included a very high number of items, similar to Baseline, resulting in low F1 scores (Table 7.6).

- **Knee:** This method identifies the "knee" or "elbow" point in the sorted similarity scores, where the rate of change in scores significantly decreases. This point is often considered an optimal balance between including enough relevant items and avoiding diminishing returns. Knee thresholding showed improved F1 scores over GMM and Otsu in our experiments, indicating a better balance.

- **Max Gap:** This method identifies the largest difference between consecutive similarity scores when sorted in descending order. The threshold is set at the score immediately preceding this largest gap, assuming it represents a clear separation between relevant and less relevant documents. In our experiments, Max Gap consistently yielded the highest F1 scores, particularly when combined with effective rerankers, by selecting a compact and highly relevant set of documents (Table 7.6).

- **Otsu:** Originally developed for image thresholding, Otsu's method can be applied to similarity scores to find a threshold that minimizes the intra-class variance of the two groups (below and above the threshold) or, equivalently, maximizes the inter-class variance. Similar to GMM, Otsu often resulted in including a large number of documents in our experiments, leading to low F1 scores.

- **2nd Derivative:** This method identifies inflection points in the sorted similarity score curve by analyzing its second derivative. A significant change in the curvature can indicate a natural cutoff point. Our experiments showed this method to perform better than simple baselines but not as effectively as Max Gap for optimizing F1 scores.

Adaptive Strategies

These strategies can combine multiple signals. For example, a system might initially retrieve a larger set of candidates and then apply a dynamic threshold based on a reranker's scores, ensuring that only the most highly relevant documents, as determined by a more sophisticated model, are passed to the LLM. The success of methods like Max Gap when combined with rerankers in our experiments highlights the power of such adaptive strategies.

3.4.3 Benefits and Challenges

The primary benefit of dynamic similarity thresholding is its ability to improve both the precision and recall of the retrieval process. By intelligently filtering out irrelevant documents, it reduces noise and the computational burden on the LLM, leading to more accurate and concise answers. Simultaneously, by allowing for a more expansive context when truly relevant documents exist, it enhances the LLM's ability to provide comprehensive responses. This adaptability makes RAG systems more robust and efficient across a wider range of queries.

However, implementing dynamic thresholding also presents challenges. Determining the "right" method for a specific dataset or application can be complex, often requiring empirical evaluation. The computational overhead of analyzing score distributions or running learned models for threshold prediction can also add latency, which needs to be balanced against the gains in retrieval quality. Careful tuning and validation are essential to ensure that dynamic thresholding genuinely enhances system performance.

3.5 Late Chunking with Contextual Chunk Embeddings

Late chunking represents an advanced strategy that fundamentally alters the generation of chunk embeddings, transitioning from isolated processing of chunks to a more holistic, context-aware methodology. As elucidated by Günther et al. (2025) [25], this technique exploits the full capacity of long-context embedding models to generate what they define as *Contextual Chunk Embeddings*.

3.5.1 Limitations of Traditional Chunking

In a conventional chunking workflow, a document is initially segmented into discrete chunks (e.g., by paragraphs, fixed token counts, per page, or via an alternative chunking strategy). Subsequently, an embedding model is applied to each chunk independently to produce its vector representation. The primary disadvantage of this method is the resultant context loss. The embedding for each chunk is generated in isolation, lacking awareness of the preceding or succeeding information within the original document. This can result in ambiguous or less informative embeddings, consequently degrading the quality of the retrieval process, as the model is unable to fully capture the semantic richness of the text.

3.5.2 The Late Chunking Process

Late chunking mitigates this limitation by inverting the process: it initially generates highly contextualized embeddings at the token level and subsequently applies chunk boundaries. The process, delineated in Algorithm 1, proceeds as follows:

1. **Tokenization and Contextualization:** Rather than initially chunking the document, the entirety of the document (or the largest feasible segment that conforms to the model's context window) undergoes tokenization. The transformer model subsequently processes this extended sequence of tokens, producing a vector representation (ω_i) for each individual token. Significantly, each of these token embeddings is context-aware, having been generated with an understanding of the entire surrounding text.

2. **Boundary Cue Application:** Following the generation of token-level embeddings ($\omega_1, \ldots, \omega_m$), the predefined chunk boundaries are applied. These boundaries, established by a standard chunking algorithm (e.g., sentence splitting), are not employed to segment the text for the model, but instead to ascertain which token embeddings correspond to specific chunks. This constitutes the pivotal step from which the technique derives its appellation: the chunking logic is applied *post-tokenization* in the process.

3. **Pooling:** Once the token embeddings for each chunk have been identified, a pooling function—typically mean pooling—is applied to the sequence of token vectors delimited by each chunk's boundaries. This process aggregates the contextualized token embeddings into a singular, final vector for each respective chunk.

This "inside-out" approach ensures that the final embedding for each chunk is not merely a representation of its internal text, but is profoundly influenced by the broader context of the entire document, thereby leading to more robust and accurate retrieval.

The concept of late chunking was pioneered by Jina AI with the introduction of their `jina-embeddings-v2` model family. It has subsequently undergone refinement and expansion in later releases, including `jina-embeddings-v3` [26] and `jina-embeddings-v4` [27]. While subsequent versions incorporated multimodal capabilities, which fall outside the scope of this investigation, the fundamental principle of late chunking for text persists as a significant innovation.

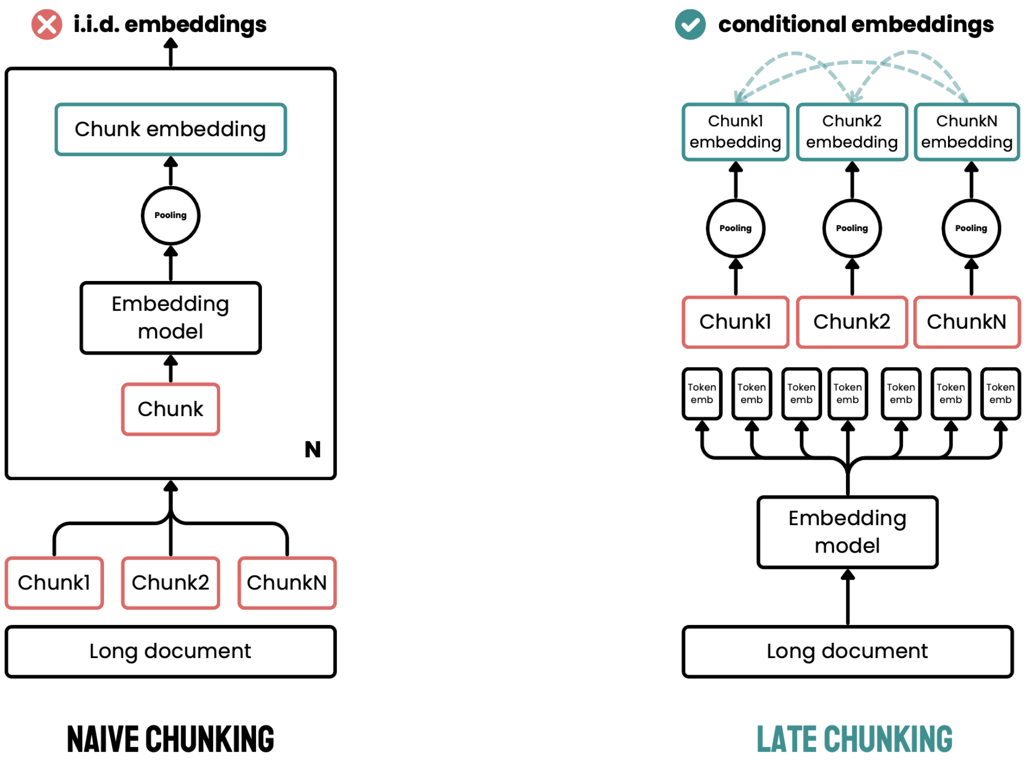


Figure 3.1: Visualization of the Late Chunking algorithm (right) compared to naive chunking (left). Image from Günther et al. (2025) [25].

Algorithm 1 Late Chunking

```
1: procedure LateChunking(document, chunk_boundaries)
2:     $tokens \leftarrow$ tokenize($document$)
3:     $\omega_1, \ldots, \omega_m \leftarrow$ Transformer($tokens$)        ▷ Generate token-level embeddings
4:     $token\_spans \leftarrow$ get_token_spans($document, tokens$)
5:     $chunk\_token\_indices \leftarrow []$
6:     for each $chunk$ in $chunk\_boundaries$ do
7:         $start\_char, end\_char \leftarrow chunk$
8:         $start\_token \leftarrow$ find_token_at_char($token\_spans, start\_char$)
9:         $end\_token \leftarrow$ find_token_at_char($token\_spans, end\_char$)
10:        append ($start\_token, end\_token$) to $chunk\_token\_indices$
11:    end for
12:    $chunk\_embeddings \leftarrow []$
13:    for each $start\_idx, end\_idx$ in $chunk\_token\_indices$ do
14:        $token\_vectors\_for\_chunk \leftarrow \omega_{start\_idx}, \ldots, \omega_{end\_idx}$
15:        $embedding \leftarrow$ MeanPool($token\_vectors\_for\_chunk$)
16:        append $embedding$ to $chunk\_embeddings$
17:    end for
18:    return $chunk\_embeddings$
19: end procedure
```

Chapter 4

Prompt Engineering for RAG

In a Retrieval-Augmented Generation system, the prompt serves as the crucial interface between the retrieved information and the generative capabilities of the Large Language Model. Effective prompt engineering is indispensable for guiding the LLM in synthesizing the provided context and generating an accurate, relevant, and coherently 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 exhibits greater complexity compared to a standard query directed at an LLM. It must effectively integrate two distinct informational components: the user's original query and the retrieved context. The structure of the prompt governs the LLM's interpretation and utilization of the provided information.

A typical RAG prompt includes:

- **Instructions:** Explicit directives to the LLM regarding its operational behavior. This may encompass instructions to exclusively utilize the provided context, to assume a particular persona, or to format the output in a specified manner.
- **Context:** The set of retrieved document chunks that are relevant to the query.
- **Query:** The user's original question or request.

The formulation of the prompt necessitates a meticulous arrangement of these elements. For instance, a typical template might be structured as follows:

```
   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

Diverse prompting styles can be employed contingent upon the specific task and the LLM in use. The selection of style can substantially influence the quality of the generated response.

- **Zero-Shot Prompting:** This represents the most prevalent style in RAG, wherein the LLM receives instructions and context without prior examples demonstrating task completion. The efficacy of zero-shot prompts is highly dependent on the clarity of the instructions and the LLM’s intrinsic capacity for reasoning and adherence to directives.
- **Few-Shot Prompting:** In this approach, the prompt incorporates a limited number of examples (shots) illustrating the desired input-output format. This can prove particularly advantageous for intricate tasks or when a highly specific output structure is mandated. For RAG, this might entail demonstrating to the model how to synthesize context into an answer for a few sample questions prior to presenting the actual query.
- **Instruction-Based Prompts:** These prompts emphasize the provision of highly detailed and explicit instructions. This may encompass 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 specific role to the LLM (e.g., "You are a financial analyst reviewing these documents") can assist in framing the context and directing the tone and focus of the response.

4.3 Prompt Evaluation

Evaluating the efficacy of different prompts constitutes a critical step in optimizing a RAG pipeline. Given that the quality of a generated response can be subjective, a combination of qualitative and quantitative methods is frequently employed.

4.3.1 Golden Datasets

A **golden dataset** comprises a curated collection of queries coupled with optimal, human-verified answers. By comparing the LLM's output for a given query against the golden answer, one can evaluate the quality of the prompt. While the creation of a golden dataset can be labor-intensive, it furnishes a robust benchmark for evaluation.

4.3.2 LLM-as-a-Judge

A more scalable approach to evaluation involves employing another, often more powerful, LLM as an evaluator, a methodology demonstrated to correlate favorably with human judgment [28]. This methodology is termed the **LLM-as-a-Judge** method. Frameworks such as **DeepEval** have materialized to standardize this process. An evaluator LLM is provided with a set of criteria and tasked with assessing the output of the RAG system across various dimensions.

4.4 Evaluation Metrics

When employing an LLM-as-a-Judge, several pivotal metrics are utilized to evaluate the quality of the RAG output. These metrics offer a multi-faceted perspective on performance:

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

- **Answer Relevancy:** Quantifies the degree to which the generated answer addresses the user's original query. A factually correct answer lacks utility if it fails to address the posed question.

- **Faithfulness:** This metric evaluates whether the LLM's response constitutes a faithful representation of the information contained within the retrieved context. A high faithfulness score indicates that the model did not fabricate information not present in the source documents.

- **Hallucination Rate:** Specifically quantifies the incidence of factually incorrect or nonsensical statements within the output. This constitutes a critical metric for constructing trustworthy RAG systems.

By systematically testing different prompt structures and styles and evaluating them against these metrics, it becomes feasible 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 functions as the generative component within the RAG system, tasked with synthesizing retrieved information and producing the final response. The selection of the generator LLM represents a pivotal decision that profoundly influences the system's overall performance, cost, and latency. This chapter presents a comparative analysis of various LLMs within the RAG context, examining how their architectural distinctions, context window capacities, and intrinsic capabilities influence their suitability for the generation task.

5.1 The Role of the Generator LLM

Within a RAG pipeline, the LLM's function is not to retrieve facts from its internal parametric memory but rather to reason upon the provided context. As emphasized by Gao et al. (2024) [2], the optimal generator LLM should demonstrate proficiency in:

- **Context Adherence (Faithfulness):** Strictly grounding its response in the provided context and precluding the introduction of extraneous information.
- **Information Synthesis:** Synthesizing facts from multiple retrieved chunks into a singular, coherent answer.
- **Instruction Following:** Precisely adhering to the instructions within the prompt, including compliance with a specified output format or persona.

- **Noise Resistance:** Disregarding irrelevant or contradictory information within the context and prioritizing the most pertinent facts.

5.2 Performance Evaluation based on Model and Context Window Size

Different LLMs demonstrate diverse levels of proficiency across these dimensions. The performance of a RAG system is consequently highly contingent upon the specific model selected for the generation step. This section assesses various models against these criteria.

5.2.1 Comparing LLM Families

LLMs can be categorized into several families, each possessing distinct characteristics. Our experiments assessed models from the following prominent families:

- **GPT (Generative Pre-trained Transformer) Series (e.g., GPT-4, GPT-4o):** Developed by OpenAI, these models are recognized for their robust reasoning and instruction-following capabilities. GPT-4, in particular, has demonstrated a high degree of faithfulness and a capacity to synthesize complex information, rendering it a favored selection for high-quality RAG systems [29].

- **Llama Series (e.g., Llama 2, Llama 3):** Developed by Meta, these constitute potent open-source models that have achieved substantial competitiveness with their proprietary counterparts. They provide a robust equilibrium between performance and customizability, thereby enabling fine-tuning on specific domains. While not assessed in our conclusive experiments, the Llama series constitutes a critical open-source alternative and a benchmark for performance within the domain [30].

- **Claude Series (e.g., Claude 3 Sonnet, Opus):** Developed by Anthropic, these models are particularly distinguished by their extensive context windows and robust performance on tasks necessitating complex reasoning and a profound comprehension of lengthy documents [31].

- **Gemini Series (e.g., Gemini 1.5 Pro, Gemini 2.5 Pro):** Developed by Google, the Gemini models are intrinsically multimodal and engineered for high performance across a broad spectrum of tasks. They have exhibited robust performance in both retrieval and generation, establishing them as a versatile option for RAG systems [32].

5.2.2 The Impact of Context Window Size

The **context window size** of an LLM delineates the maximum volume of text (prompt + retrieved context + generated response) that the model can process concurrently. This carries several implications for RAG, as elucidated by Gao et al. (2024) [2]:

- **Information Density:** A larger context window facilitates the inclusion of a greater number of retrieved chunks to the LLM, potentially enhancing the comprehensiveness of the generated answer. However, this concurrently elevates the risk of the "lost in the middle" problem, wherein the model may disregard pertinent information embedded within an extensive context [6].
- **Cost and Latency:** Processing larger contexts incurs greater computational expense, resulting in elevated operational costs and prolonged response times.
- **Architectural Differences:** Newer models featuring exceptionally large context windows (e.g., Claude 3) are engineered to more effectively locate and utilize information within extensive documents, which can confer a substantial advantage for specific RAG applications.

5.3 Trade-offs in LLM Selection

Selecting the appropriate LLM for a RAG system necessitates balancing several factors:

- **Performance:** The quality of the generated output, quantified by metrics such as faithfulness, relevancy, and accuracy.
- **Cost:** The financial cost per generated token, which can fluctuate considerably among models.
- **Speed (Latency):** The latency incurred by the model in generating a response.
- **Customizability:** The capacity to fine-tune the model on domain-specific data, a process frequently more straightforward with open-source models.

Ultimately, the optimal selection is contingent upon the specific requirements of the application. A customer-facing chatbot might prioritize rapid response and low operational cost, whereas a legal research assistant would prioritize maximal 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 represents a synthesis of information derived from multiple retrieved document chunks. For the purposes of transparency, debuggability, and continuous improvement, it is imperative to ascertain precisely which segments of the retrieved context contributed to the constructed answer. This chapter investigates the methodologies and significance of analyzing the alignment between the generated response and the source chunks, a process frequently termed *citation and attribution* or *groundedness*.

6.1 The Importance of Chunk-Response Alignment

Comprehending the nexus between the source context and the final output, as underscored by Gao et al. (2024) [2], fulfills several pivotal functions:

- **Trust and Verifiability:** For users, particularly in critical applications such as medical or legal research, the ability to verify the source of a particular statement is essential for establishing trust in the system's output. Citations enable users to independently verify the information.

- **Debugging and Evaluation:** When a RAG system generates a suboptimal or erroneous answer, tracing the response back to the source chunks constitutes the initial step in problem diagnosis. It assists in determining whether the issue resides with the retriever (e.g., retrieving irrelevant context), the generator (e.g., misinterpreting the context), or the source documents themselves.

- **System Improvement:** By analyzing which chunks are consistently utilized to address specific types of questions, insights can be gained into the performance of the retrieval system. If high-quality chunks are being disregarded or low-quality chunks are being employed, it may signal a necessity to fine-tune the embedding model or modify the reranking strategy.

- **Feedback Loops:** In advanced RAG systems, the identification of salient chunks can furnish a feedback signal to the retriever, enabling it to adapt and enhance its performance over time through reinforcement learning or other adaptive methodologies.

6.2 Methods for Analyzing Chunk-Response Alignment

Various techniques can be employed to trace the provenance of the information within the generated response. The complexity and accuracy of these methods can fluctuate considerably, and they frequently entail a trade-off between computational cost and precision.

6.2.1 Prompt-Based Attribution

The most straightforward method involves explicitly instructing the LLM to cite its sources within the prompt. The instructions may incorporate a directive such as: *"After each sentence in your response, cite the ID of the source document you used to formulate that sentence."*

While straightforward, this approach possesses inherent limitations. The LLM may not consistently adhere to the instructions, and it can occasionally hallucinate citations or erroneously attribute information. The reliability of this method is highly contingent upon the instruction-following capabilities of the selected LLM.

6.2.2 Post-Hoc Similarity Analysis

An alternative approach involves analyzing the alignment subsequent to the generation of the response. This can be achieved by:

1. Decomposing the generated response into individual sentences or claims.

2. For each sentence, computing its embedding.

3. Comparing the embedding of the generated sentence against the embeddings of the original retrieved chunks.
4. The chunk exhibiting the highest semantic similarity to a given sentence is deemed its most probable source.

This method offers a more quantitative and verifiable means of linking the output to the input, though it is not infallible. A generated sentence may synthesize information from multiple chunks, rendering a one-to-one mapping challenging.

6.2.3 Analyzing Attention Mechanisms

For transformer-based LLMs, the internal *attention mechanism* can theoretically offer insights into which components of the input context exerted the most influence in generating a particular segment of the output. By inspecting the attention weights, one can discern which of the retrieved chunks the model prioritized during the generation of a specific word or phrase.

In practice, this constitutes a highly complex approach. Accessing and interpreting attention weights can be challenging, particularly with proprietary, black-box models. Furthermore, the correlation between high attention scores and factual contribution is not invariably direct and remains an active area of research.

6.2.4 Building a Knowledge Graph

A more structured approach entails constructing a knowledge graph from the source documents. The graph would comprise entities and their interrelationships. Upon generation of a response, the entities referenced therein can be linked back to the nodes within the knowledge graph, thereby providing a clear and structured form of attribution. This constitutes a potent yet resource-intensive method that is optimally suited for well-defined domains.

6.3 Evaluation Metrics for Retrieval

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

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

- **Recall@k:** The proportion of all relevant documents within the corpus that are identified among the top-k retrieved documents.
- **Mean Reciprocal Rank (MRR):** The mean of the reciprocal ranks of the first relevant document for a given set of queries. A higher MRR signifies that the system demonstrates superior capability in ranking relevant documents more prominently.
- **Normalized Discounted Cumulative Gain (nDCG):** A metric of ranking quality that accounts for 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 delineates the experimental methodologies and presents the findings from a series of evaluations conducted to assess various components and strategies within Retrieval-Augmented Generation systems. The experiments encompass a spectrum of techniques, ranging from fundamental retrieval and chunking methods to advanced reranking, prompt engineering, and the influence of different embedding and generative models. The overarching objective 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 employing a consistent setup.

7.1.1 Dataset(s)

The primary dataset utilized for these experiments comprises a curated collection of academic papers and articles pertaining to the field of artificial intelligence. This dataset was selected due to its inherent complexity, specialized technical vocabulary, and the requirement for precise, fact-based answers. For question generation and evaluation, a set of 100 questions was meticulously crafted, encompassing a range of topics within the dataset. Each question is formulated to possess a verifiable answer within the document corpus.

7.1.2 Baseline Configuration

To quantify the impact of various optimization techniques, a baseline RAG configuration was established:

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

7.1.3 Core Evaluation Metrics

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

- **Retrieval Performance:** Evaluated using *Precision@k*, *Recall@k*, and *Mean Reciprocal Rank (MRR)*. These metrics are paramount for comprehending the effectiveness of the retrieval stage in isolation.
- **Generation Quality:** Assessed using the *LLM-as-a-Judge* method with the DeepEval framework. The primary metrics tracked were *Faithfulness*, *Answer Relevancy*, and *Hallucination Rate* [28].

7.2 Experiment 1: Embedding Models

This experiment assesses the performance of various embedding models on long-context retrieval tasks. The objective is to comprehend how diverse architectural and chunking strategies influence retrieval performance on documents of varying lengths and complexity.

7.2.1 Models and Methodology

We compared five embedding models on the **LongEmbed** benchmark [33], a suite of datasets formulated to evaluate long-context retrieval. The models assessed were:

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

- **Nomic AI `nomic-embed-text`:** An open-source model possessing a vector size of 768. This model employs a prefix for distinct 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 [34], 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 comprises six datasets, two of which are synthetic and four are derived from real-world data. The datasets are:

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

- **NarrativeQA:** A question-answering dataset featuring extended narratives.

- **Needle In A Haystack (NIAH):** A synthetic dataset designed to evaluate the model's capacity to locate a specific piece of information ("needle") in an extensive text ("haystack").

- **Passkey Retrieval:** A synthetic dataset designed to assess the model's ability to retrieve a specific key from an extended document.

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

- **SummScreenFD:** A summarization dataset derived from television show scripts.

7.2.3 Results and Discussion

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

The Rise of Compact, High-Performance Models While our results identify **`Qwen3-Embedding-0.6B`** as the leading performer, its true significance becomes apparent when contextualized within the global Massive Text Embedding Benchmark (MTEB) leaderboard. As of August 1st, 2025, this model occupies an impressive 4th rank overall, surpassed solely by Google's proprietary `gemini-embedding-001` and its own, substantially larger 4B and 8B parameter siblings [19]. With merely 595M parameters, its performance is noteworthy, demonstrating that architectural innovations and advanced training pipelines can enable smaller, open-source models to surpass the performance of significantly larger competitors. This efficiency stems directly from its foundation on the Qwen3 architecture [35] and a sophisticated multi-stage training process that leverages the foundational LLM itself to synthesize high-quality training data [36].

A New Baseline: The Obsolescence of Ada-002 For years, OpenAI's **`ada-002`** served as a de facto industry standard. However, these results, coupled with its MTEB ranking of 167th, confirm that it has been decisively superseded. Newer models, leveraging innovative architectures and specialized training techniques, now provide substantially superior performance. The other models in our evaluation also rank significantly higher than `ada-002` on the MTEB leaderboard: `jina-embeddings-v3` ranks 24th, `nomic-embed-text` ranks 54th, and even the earlier `jina-embeddings-v2-base-en` ranks 154th [19]. This trend indicates that the benchmark for state-of-the-art has shifted towards more open and efficient models.

The Enduring Value of Specialization Despite the prevalence of a robust generalist model like Qwen3, our findings also emphasize the persistent importance of specialized models. **`jina-embeddings-v3`**, with its unique late chunking strategy, maintains its position as the leading performer on the `passkey` task, thereby validating the efficacy of its architectural design for retrieving specific facts from structured long documents. Similarly, **`nomic-embed-text`**'s superior performance on the `needle` dataset affirms its designation as a premier model for extreme "needle-in-a-haystack" scenarios. This demonstrates that for specific, challenging use cases, a specialized model can remain the optimal choice.

Distance Metrics and Model Compatibility For most models, the selection between L2 and Inner Product (IP) distance metrics yields comparable results, indicating that their embeddings are adequately normalized. A crucial observation, however, pertains to the performance degradation of **`jina-embeddings-v2-base-en`** when employing the IP metric. As observed across all tables, its performance significantly diminishes with IP; for instance, its average Recall@5 decreases from 51.85% (L2) to a mere 12.12% (IP). This underscores that its embedding vectors are not normalized for cosine similarity, rendering L2 distance the sole viable choice for this model and emphasizing the importance of aligning the retrieval metric with the model's output properties.

Conclusion In summary, this experiment exemplifies 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, indicates a shift towards more efficient, accessible, and open-source solutions that attain state-of-the-art results.

- Models such as **`ada-002`**, while historically significant, are now considered obsolete. They have been superseded by a new generation of models featuring more advanced architectures, training methodologies, and specialized strategies such as late chunking.

- While general-purpose models have improved substantially, **specialization remains highly valuable**. For niche but critical tasks such as extreme long-context fact retrieval, models like `jina-embeddings-v3` and `nomic-embed-text` offer targeted performance that can surpass even the most proficient generalists.

- **Technical implementation details are paramount**. The selection of the distance metric must be compatible with the model's properties, as the suboptimal performance of `jina-embeddings-v2-base-en` with the IP metric illustrates.

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
	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	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
5	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**
	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
10	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**
	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
25	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
	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	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
5	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**
	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
10	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**
	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
25	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
	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	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
5	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**
	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
10	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**
	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
25	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 assessed the benefit of incorporating a reranking step subsequent to the initial retrieval. A reranker, typically a cross-encoder model, re-evaluates the top-k documents returned by the initial retriever, thereby furnishing a more accurate relevance score. This facilitates the recovery 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 yielded by the similarity search:

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

- **GTE ML Reranker Base:** A robust reranker based on the DeBERTa-v3 architecture, recognized for its strong performance on various reranking tasks [37].
- **Jina Reranker v1 Tiny EN:** A lightweight and efficient reranker from Jina AI, engineered for rapid processing and scalability [38].
- **MXBAI Rerank Base v2:** A multilingual reranker that has demonstrated competitive performance on diverse datasets [39].

7.3.2 Results and Discussion

The results presented in Table 7.6 and the visualizations in Figure 7.1 unveil a complex interplay among reranking models, chunking strategies, and thresholding methods. The principal finding is that the effectiveness of a reranker is not absolute but is highly contingent on the subsequent thresholding strategy employed to select the final documents.

When no intelligent thresholding is applied (Baseline) or when suboptimal methods such as GMM or Otsu are utilized, the performance gains from reranking are minimal or even negative. These methods frequently include an excessive number of documents (as observed in Table 7.7), thereby 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 due to the high recall being compromised by extremely low precision.

However, the full potential of reranking is realized when combined with an effective thresholding method such as **Max Gap**. Utilizing this method, the **GTE ML Reranker Base** model attains the highest F1 score (**0.654** for Full Page, **0.646** for Page Split), thereby demonstrating a substantial improvement over the No Reranker baseline (0.456 and 0.598). This is attributable to the Max Gap method's high efficacy in identifying the point of diminishing returns within the reranked list, enabling the selection of a compact, highly relevant set of documents. As depicted in Table 7.7, the GTE reranker with Max Gap processes an average of only 4.76 items, resulting in a high precision of 0.594.

Conversely, the **MXBAI Rerank Base v2** model exhibits suboptimal performance with aggressive thresholding methods such as Knee and Max Gap. This is due to the model's score distribution being notably flat, which impedes these methods from identifying a distinct cutoff point. Consequently, it processes an insufficient number of documents (e.g., only 4.38 items with Knee), leading to a significant

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

reduction in recall (0.388) and a low F1 score (0.105).

In conclusion, this experiment demonstrates that merely incorporating a reranker is insufficient. The combination of a robust reranker such as **GTE ML Reranker Base** with an aggressive, well-suited thresholding method like **Max Gap** yields optimal performance, maximizing precision without compromising essential recall.

7.4 Experiment 3: Prompt Engineering to Prevent Hallucination

This experiment centers on the generation component of the RAG pipeline, assessing a broad spectrum of Large Language Models to function as the generator. The primary objective was to identify the optimal-performing model and the most suitable prompt and temperature configuration for our specific use case.

7.4.1 Motivation and Baseline

The evaluation commenced with a baseline configuration of **GPT-4o** employing prompt `P1` and a temperature of `0.2`. As novel and more powerful models were introduced during the course of this research, they were systematically evaluated against this baseline. This continuous evaluation process enabled us to remain at the forefront of research and select the most suitable model for integration into a production RAG system, thereby ensuring that any replacement would provide superior performance.

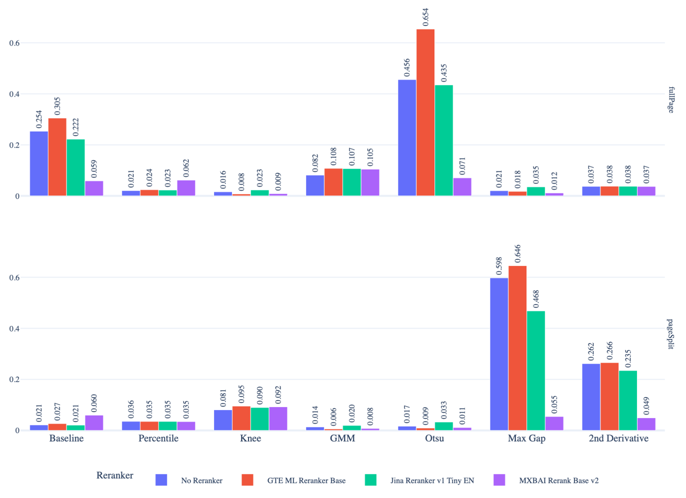
7.4.2 Evaluation Methodology

We utilized the **DeepEval** framework [40], which employs an LLM-as-a-Judge approach to assess the generator's output. For subjective, use-case-specific evaluations, we employed DeepEval's **G-Eval** functionality, which draws inspiration from the G-Eval framework [41] and utilizes a chain-of-thought process to evaluate outputs against custom criteria. We delineated two such metrics:

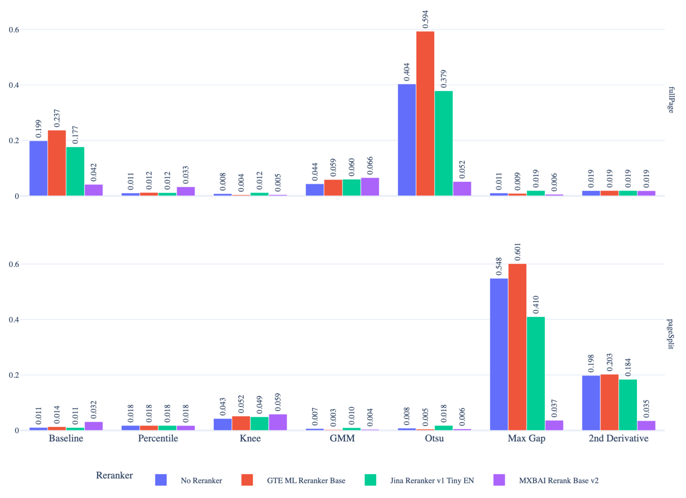
- **Correctness (C):** This metric ascertains whether the actual output is factually correct based on the expected output. It heavily penalizes the omission of crucial details and the inclusion of information that contradicts the expected output.

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.

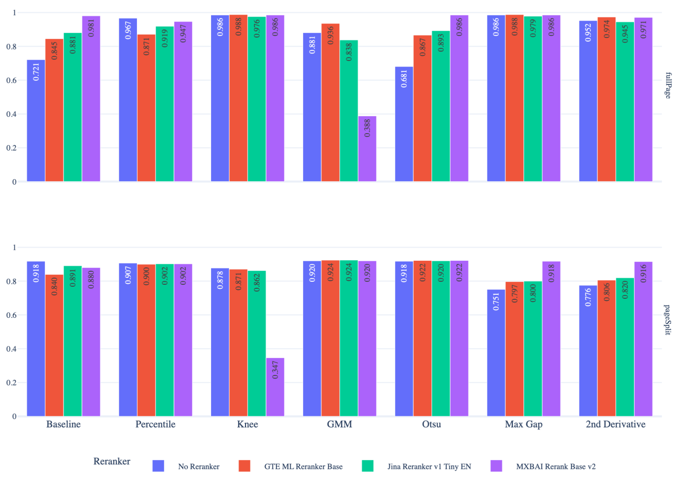
	Chunking	Full Page		Page Split	
		Docs Passed	Items Passed	Docs Passed	Items Passed
Threshold	Model				
Baseline	No Reranker	7.13	104.00	5.43	101.14
	GTE ML Reranker Base	6.28	79.70	5.46	68.79
	Jina Reranker v1 Tiny EN	7.13	94.76	5.91	103.57
	MXBAI Rerank Base v2	6.40	102.88	5.42	109.06
GMM	No Reranker	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



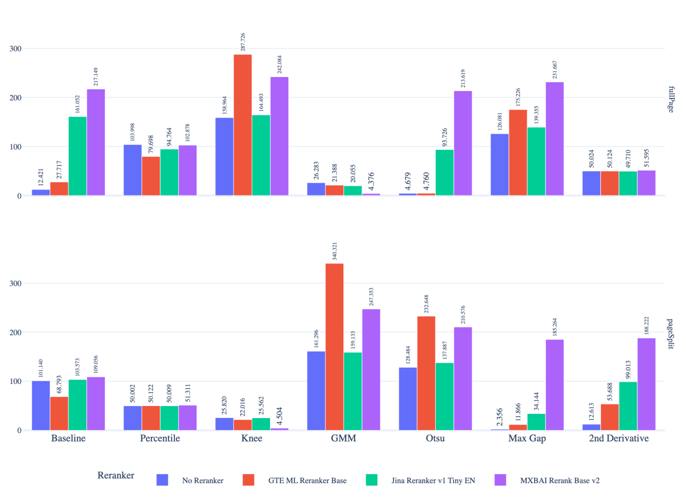
(a) Average F1 score by facet



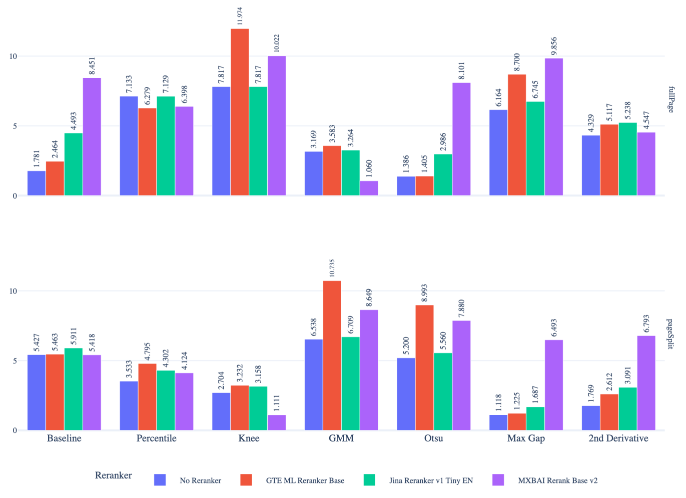
(b) Average precision by facet



(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.

- **Specific Information Accuracy (SIA):** This metric evaluates whether the model's response appropriately utilizes information from the context without introducing specific details (names, places, numbers) that are not explicitly provided. It is particularly important for assessing the model's capacity to discern when a question cannot be answered from the provided context. For example, a high SIA score is conferred if the model accurately indicates its inability to answer a question such as "Who is Cinderella?" when presented with an anonymized version of the story, as this demonstrates its non-reliance on its internal parametric knowledge.

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

- **Answer Relevancy (AR):** This metric quantifies the relevance of the generated output to the user's input query. It ensures that the model's response is pertinent and directly addresses the posed question [40].
- **Faithfulness (F):** This metric evaluates whether the generated output factually corresponds with the information present in the retrieved context. A high score indicates that the model did not fabricate facts and adhered faithfully to the source material [40].
- **Hallucination (H):** This metric ascertains if the model generates factually incorrect information by comparing the output to the provided context. It is a critical measure for ensuring the trustworthiness of the RAG system [40].

7.4.3 Results and Discussion

The comprehensive results are delineated in Table 7.8. The baseline configuration (GPT-4o, P1, Temp 0.2) yielded a total score of 370.52. The results indicate that several newer models and prompt configurations were capable of significantly outperforming this baseline.

For instance, the **O4 Mini** model employing prompt `P2` and a temperature of `0.2` attained the highest total score of **403.86**. This demonstrates the utility of continuous evaluation, as it enabled us to identify a model that not only performs superiorly to the original baseline but also to ascertain the optimal prompt (`P2`) and temperature (`0.2`) for it. These findings are crucial for deploying a RAG system that is not only accurate and faithful but also robust in addressing 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] Vladimir Karpukhin et al. “Dense Passage Retrieval for Open-Domain Question Answering”. In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 2020, pp. 6717–6731.

[4] Kelvin Guu et al. “REALM: Retrieval-Augmented Language Model Pre-Training”. In: *Proceedings of the 37th International Conference on Machine Learning (ICML)*. 2020, pp. 3849–3859.

[5] Wenhao Yu et al. “A Survey of Knowledge-Enhanced Text Generation”. In: *ACM Computing Surveys* 55.1 (2022), pp. 1–40.

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

[7] 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`.

[8] Gerard Salton, Anita Wong, and Chung-Shu Yang. “A vector space model for automatic indexing”. In: *Communications of the ACM* 18.11 (1975), pp. 613–620.

[9] 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`.

[10] 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`.

[11] Martin Aumüller, Erik Bernhardsson, and Alexander Faithfull. *ANN-Benchmarks: A Benchmarking Tool for Approximate Nearest Neighbor Algorithms.* 2018. arXiv: `1807.05614 [cs.IR]`. URL: `https://arxiv.org/abs/1807.05614`.

[12] Yanzhao Zhang et al. “The Impact of Chunking Strategies on Retrieval-Augmented Generation”. In: *arXiv preprint arXiv:2312.09224* (2023).

[13] LangChain Authors. *LangChain: Building applications with LLMs through composability.* `https://github.com/langchain-ai/langchain`. Accessed: 2025-08-31. 2022.

[14] Steven Bird, Ewan Klein, and Edward Loper. *Natural Language Processing with Python.* O’Reilly Media, Inc., 2009.

[15] Matthew Honnibal and Mark Johnson. “SpaCy: Industrial-strength Natural Language Processing in Python”. In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing: System Demonstrations.* 2015, pp. 1–4.

[16] Jacob Devlin et al. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”. In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers).* 2019, pp. 4171–4186.

[17] Yinhan Liu et al. “RoBERTa: A Robustly Optimized BERT Pretraining Approach”. In: *arXiv preprint arXiv:1907.11692* (2019).

[18] Yanzhao Chen et al. “Improving Retrieval-Augmented Generation with Domain-Specific Embeddings”. In: *arXiv preprint arXiv:2310.07007* (2023).

[19] 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).

[20] Stephen E Robertson et al. “Okapi at TREC-3”. In: *Proceedings of the third Text REtrieval Conference (TREC-3).* 1995, pp. 109–126.

[21] Yingqi Qu et al. *RocketQA: An Optimized Training Approach to Dense Passage Retrieval for Open-Domain Question Answering.* 2021. arXiv: `2010.08191 [cs.CL]`. URL: `https://arxiv.org/abs/2010.08191`.

[22] Rodrigo Nogueira and Kyunghyun Cho. *Passage Re-ranking with BERT.* 2020. arXiv: `1901.04085 [cs.IR]`. URL: `https://arxiv.org/abs/1901.04085`.

[23] 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`.

[24] Yunchang Zhu et al. *Adaptive Information Seeking for Open-Domain Question Answering.* 2021. arXiv: `2109.06747 [cs.CL]`. URL: `https://arxiv.org/abs/2109.06747`.

[25] 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`.

[26] 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`.

[27] 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`.

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

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

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

[31] Anthropic. *The Claude 3 Model Family: Opus, Sonnet, Haiku.* `https://www.anthropic.com/news/the-claude-3-model-family-opus-sonnet-haiku`. Accessed: 2025-08-01. 2024.

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

[33] 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.

[34] 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.

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

[36] 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.

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

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

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

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

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