Evaluating Knowledge Base implemenations for Retrieval Augmented Generation

Third Year Project Report

Bradley Bell
10829685
Supervised by Dr Chenghua Lin

Abstract

Retrieval-Augmented Generation (RAG) promises to anchor Large Language Models (LLMs) to factual reality, yet this promise is frequently bottlenecked by the architecture of the external Knowledge Base (KB). The optimal strategy for structuring this KB, particularly for complex and evolving technical documentation, remains a critical question. This project confronts this challenge through an empirical investigation into the performance of three distinct KB structures: naive Monolithic, Chunked, and an Enhanced approach integrating semantic metadata directly into vector representations. Using the HarmonyOS ArkTS developer documentation as a testbed, we implemented a consistent RAG pipeline featuring two-stage retrieval. Performance was assessed using a multi-faceted evaluation strategy, combining precision-focused ground-truth metrics derived from human validation, scalable LLM-as-judge assessments of context and answer quality, and essential operational benchmarks (latency, token cost). **The findings are decisive: structured KBs (Chunked and Enhanced) massively outperform the Monolithic approach across key retrieval metrics.** While metadata enhancement provides marginal gains in specific scenarios, standard Chunking demonstrates robust performance, particularly for complex, multi-document queries. This work delivers evidence on KB structuring trade-offs, providing actionable insights for engineering RAG systems in knowledge-intensive technical domains.

Contents

List of Figures

List of Tables

Abbreviations

ANN	Approximate Nearest Neighbor
API	Application Programming Interface
BFS	Breadth-First Search
CI	Continuous Integration
CoT	Chain-of-Thought
CPU	Central Processing Unit
GT	Ground Truth
HTML	HyperText Markup Language
IDCG	Ideal Discounted Cumulative Gain
IR	Information Retrieval
JS	JavaScript
JSON	JavaScript Object Notation
KB	Knowledge Base
LLM	Large Language Model
MRR	Mean Reciprocal Rank
NDCG	Normalized Discounted Cumulative Gain
NLP	Natural Language Processing
P@k	Precision at k
R@k	Recall at k

RAG	Retrieval-Augmented Generation
RQ	Research Question
TF-IDF	Term Frequency-Inverse Document Frequency
TREC	Text REtrieval Conference
UML	Unified Modeling Language
URL	Uniform Resource Locator

Chapter 1

Introduction

1.1 The Imperative for Grounded Knowledge in LLMs

Large Language Models (LLMs), powered by architectures like the Transformer [Vaswani et al., 2017], represent a leap forward in artificial intelligence, demonstrating profound fluency in processing and generating human language. Their capabilities stem from knowledge implicitly encoded during pre-training on vast datasets. Yet, this very reliance on internalized, static knowledge creates inherent vulnerabilities. LLMs remain tethered to their last training date, rendering them obsolete for domains demanding up-to-the-date accuracy – a fatal flaw when dealing with rapidly evolving technical documentation, APIs, and software frameworks. Furthermore, their propensity for "hallucination" – confidently generating plausible but factually erroneous statements – and their often-superficial grasp of specialized domains severely limits their reliability in high-stakes applications where precision is non-negotiable.

Retrieval-Augmented Generation (RAG) [Lewis et al., 2020] offers a compelling solution. By dynamically retrieving relevant information from an external Knowledge Base (KB) before generation, RAG systems aim to ground LLM outputs in verifiable, contextually appropriate facts. This mechanism holds the potential to drastically reduce fabrications, inject current information, and provide the domain-specific depth that LLMs inherently lack. However, the realization of this potential hinges critically on the KB itself. A poorly structured or indexed KB can cripple the RAG system, feeding the LLM irrelevant or incomplete information, thereby undermining the entire process. How, then, should we structure this crucial knowledge source, especially when dealing with the intricate, interconnected, and constantly updated information typical of technical documentation like the HarmonyOS ArkTS guide – a representative example of the challenges faced?

1.2 The Knowledge Base Structuring Dilemma

While the RAG paradigm is established, the optimal strategy for structuring the underlying Knowledge Base remains a significant open question demanding rigorous empirical investigation. The choices are consequential: Should entire documents be treated monolithically, preserving original structure at the risk of exceeding context limits and diluting vector representations? Does segmenting documents into semantic chunks reliably improve retrieval focus? Can enriching these chunks by embedding metadata directly into the vector space yield a more nuanced semantic search, or does it merely introduce noise?

The lack of definitive, comparative evidence forces RAG system designers to rely on heuristics or intuition when making these critical architectural decisions. This is particularly problematic for technical domains where accuracy is paramount. A suboptimal KB structure can lead to inaccurate information retrieval, flawed code generation, wasted developer time, and ultimately, a failure to deliver on the promise of reliable AI assistance. Addressing this requires moving beyond isolated case studies to a systematic, quantitative comparison of KB structuring strategies under realistic conditions.

1.3 Aims and Objectives

The primary aim of this research is to deliver a rigorous, comparative evaluation of distinct Knowledge Base structuring strategies, quantifying their impact on the end-to-end performance of a Retrieval-Augmented Generation system operating on complex technical documentation.

To realize this aim, the project pursued the following core objectives:

- **Data Foundation:** Systematically acquire and structure the official HarmonyOS ArkTS developer documentation via an adaptable, intelligent web scraping process.
- **KB Architecture Implementation:** Construct three KBs representing distinct structural philosophies from the processed data:
  - *Monolithic:* Baseline using whole documents.
  - *Chunked:* Standard approach using semantically segmented document chunks.
  - *Enhanced:* Advanced approach embedding curated metadata alongside chunk content before vectorization.

- **RAG Pipeline Construction:** Engineer a consistent, modular RAG pipeline employing two-stage retrieval (vector search + cross-encoder re-ranking) adaptable to each KB format.
- **Rigorous Evaluation Framework:** Design and deploy a multi-faceted evaluation suite, crucially incorporating:
  - Semi-automated, human-validated ground truth creation for objective retrieval assessment.
  - Scalable LLM-as-judge metrics for nuanced context and answer quality analysis.
  - Operational performance tracking (latency, token usage).
- **Comparative Analysis & Insight Generation:** Execute the RAG pipeline across all KB formats and systematically analyze performance using the evaluation framework to determine the efficacy and trade-offs of each structuring strategy, thereby directly answering the core research questions.

1.4 Research Questions

The primary aim of this research translates into the following research questions (RQs):

1. **RQ1:** How does the structure of the Knowledge Base (monolithic vs. chunked vs. enhanced chunked) impact the relevance and accuracy of information retrieved and generated by a RAG system for ArkTS development queries?
2. **RQ2:** What is the trade-off between retrieval/generation performance and computational overhead (generation time, query latency) for each KB structure?
3. **RQ3:** How effectively can an enhanced chunked KB, incorporating metadata like keywords and summaries, improve RAG performance compared to a standard chunked approach?

1.5 Scope and Delimitations

This investigation centers on comparing the Monolithic, Chunked, and specific metadata-Enhanced chunking strategies using the HarmonyOS ArkTS documentation dataset, processed via a defined scraping methodology, and evaluated with a

specific RAG pipeline configuration (BGE-base-en-v1.5 embeddings, ChromaDB, cross-encoder/ms-marco-MiniLM-L-6-v2 re-ranker, GPT-4-Turbo/DeeSeek generation). The study does not exhaustively explore all possible chunking algorithms, metadata integration techniques (e.g., separate metadata filtering), or other advanced RAG variations (e.g., query expansion, graph-based RAG). Findings are interpreted within this defined scope. The ground truth evaluation, while carefully constructed, covers a representative subset of the generated test questions due to the inherent cost of human annotation.

1.6 Evaluation Strategy Justification

Evaluating RAG systems comprehensively presents unique challenges, particularly the "Ground Truth Scarcity Problem" when dealing with large, domain-specific corpora like the one used here. Obtaining exhaustive, human-verified relevance judgments for thousands of documents against hundreds of queries is practically infeasible. Simultaneously, relying solely on automated LLM-as-judge metrics carries risks of inherent biases and potential misalignment with human notions of quality.

Therefore, a multi-faceted evaluation strategy was not just beneficial, but essential for drawing credible conclusions. This project deliberately combines:

- **Ground-Truth Retrieval Metrics (P@k, R@k, MRR, NDCG@k):** Provide objective, high-fidelity assessment of raw retrieval effectiveness against human judgment for a core question subset. This anchors the evaluation in verifiable relevance.
- **LLM-Evaluated Quality Metrics:** Offer scalable proxies for assessing the semantic quality of retrieved context and the generated answer's relevance, faithfulness, and completeness across the entire question set. This provides breadth where exhaustive human judgment is impossible.
- **Operational Metrics:** Quantify the practical costs (latency, token usage) associated with each KB structure, crucial for assessing real-world deployability and efficiency trade-offs.

By triangulating insights from these complementary metric types, this strategy aims to provide a robust, nuanced, and pragmatically grounded comparison of the KB structuring approaches, acknowledging the limitations of any single evaluation method while maximizing the reliability of the overall findings.

1.7 Structure of the Thesis

The remainder of this dissertation is structured as follows:

- Chapter 1 introduces the research problem, motivation, aims, objectives, research questions, scope, and evaluation strategy.
- Chapter 2 reviews the background literature on RAG, knowledge base design, and evaluation techniques relevant to this work.
- Chapter 3 details the systematic methodology employed for data acquisition, KB construction rationale, and RAG pipeline design.
- Chapter 4 provides specific implementation details for the key components developed, such as the web scraper, KB transformer, and RAG pipeline modules.
- Chapter 5 presents the detailed evaluation framework implementation and analyzes the empirical results obtained from executing it across the different KB structures.
- Chapter 6 synthesizes the key findings, discusses contributions, reflects on limitations, and proposes directions for future research.

Chapter 2

Background and Related Work

The Retrieval-Augmented Generation (RAG) paradigm sits at the intersection of Information Retrieval (IR) and Natural Language Generation (NLG), aiming to leverage the strengths of both fields to produce more accurate, timely, and verifiable responses from Large Language Models (LLMs). This chapter explores the foundational concepts, common techniques, and evaluation challenges relevant to this project.

2.1 The Imperative for Knowledge Grounding in Large Language Models

The advent of Large Language Models (LLMs), largely propelled by the Transformer architecture [Vaswani et al., 2017], has marked a significant milestone in artificial intelligence. These models demonstrate impressive capabilities in natural language understanding and generation, deriving this fluency from vast amounts of knowledge implicitly encoded within their parameters during pre-training on extensive text corpora [Matarazzo and Torlone, 2025, Lan et al., 2024, Cao et al., 2024]. However, this reliance on static, internal **parametric memory** introduces fundamental limitations that challenge their deployment in knowledge-intensive or high-stakes applications.

Three primary limitations necessitate external knowledge integration. First, **knowledge cutoffs** restrict LLMs to information available up to their last training date [Lan et al., 2024, Cao et al., 2024, Prompting Guide, nodate]. This inherent staleness makes them unsuitable for tasks requiring current information, especially in dynamic fields such as technical documentation where frameworks and APIs evolve rapidly [Cao et al., 2024]. The static nature of parametric knowledge means models cannot reason about or access real-time events post-training. Second, LLMs are susceptible to **hallucinations**, generating outputs that appear

coherent and fluent but are factually incorrect or nonsensical [Wang et al., 2025, Lan et al., 2024, Ahmed et al., 2024]. This tendency, potentially linked to training data artifacts, the opaqueness of their internal reasoning, or the autoregressive generation process prioritizing plausible language over factual accuracy [Wang et al., 2025, Ahmed et al., 2024], poses significant reliability risks, particularly when models present incorrect information with high confidence [Dahl et al., 2024]. Verifying the factual basis of their outputs can be challenging. Third, despite broad training, LLMs often exhibit **domain-specificity gaps**, lacking the deep, specialized knowledge required for nuanced reasoning within complex fields [Lan et al., 2024, Cao et al., 2024]. They may struggle with specific jargon or fail to capture the intricacies of specialized domains not well-represented in their generalist training data, impacting the quality and relevance of their responses on specialized topics.

Retrieval-Augmented Generation (RAG) emerged as a prominent paradigm specifically designed to address these shortcomings [Lan et al., 2024, Cao et al., 2024, Lewis et al., 2020]. RAG frameworks augment the generative capabilities of LLMs by incorporating a mechanism to retrieve relevant information from external, **non-parametric knowledge** sources at inference time. This retrieved context allows the LLM to ground its output in specific, verifiable, and potentially up-to-date factual information, thereby mitigating hallucinations, overcoming knowledge cutoffs, and enabling adaptation to specialized domains [Lewis et al., 2020, Lan et al., 2024, Cao et al., 2024]. The core principle involves leveraging the complementary strengths of the LLM's broad linguistic competence (parametric memory) and the external source's specific knowledge (non-parametric memory) [Lan et al., 2024, Cao et al., 2024, Lewis et al., 2020, Prompting Guide, nodate].

2.2 Fundamental RAG Architecture

Conceptually, RAG systems typically operate via a two-stage process involving distinct retriever and generator components [Lan et al., 2024, Cao et al., 2024, Verma et al., 2023]. The **Retriever** is tasked with searching an external knowledge base (often a large corpus of documents indexed for efficient access) to identify information pertinent to an input query [Lewis et al., 2020, Bao et al., 2024]. Modern approaches heavily favor dense retrieval, utilizing embeddings to capture semantic meaning [Lan et al., 2024, Verma et al., 2023, Jiang et al., 2024]. The **Generator**, usually a powerful pre-trained LLM, then receives the original query alongside the retrieved contextual information and synthesizes the final output, aiming for coherence and faithfulness to the provided evidence [Cao et al., 2024, Bao et al., 2024, Sun et al., 2025].

The standard workflow follows a "retrieve-then-generate" pattern [Microsoft Learn, nodate, Verma et al., 2023]: an input query triggers the retriever to fetch

relevant passages (often the top-K most similar chunks), which are then formatted and prepended to the query to form an augmented prompt for the LLM generator [Sun et al., 2025, Jiang et al., 2024, Li et al., 2024b, Zhang et al., 2025c]. While foundational work like Lewis et al. (2020) established this core architecture [Lewis et al., 2020], the field continues to evolve towards more sophisticated and modular designs to tackle the inherent complexities [Verma et al., 2023].

2.3 Vector Embeddings: The Language of Semantic Similarity

Vector embeddings are central to modern RAG retrieval mechanisms. They represent text (queries, documents, chunks) as dense numerical vectors in a high-dimensional space [Cao et al., 2024, Reimers and Gurevych, 2019]. Specialized embedding models, often based on transformer architectures like BERT [Reimers and Gurevych, 2019]or tailored models like BGE, learn these representations by processing vast amounts of text. During training (often involving tasks like predicting context words or determining if sentences follow each other), these models learn to place texts with similar meanings closer together in the vector space, while dissimilar texts are placed further apart.

This proximity enables **semantic search**: finding documents relevant in meaning to a query, even if they use different terminology [Li et al., 2025]. The power of embeddings lies in their ability to represent these nuanced semantic relationships, allowing retrievers to identify contextually relevant information more effectively than traditional sparse methods like TF-IDF for many query types. Similarity between these dense vectors is typically quantified using metrics like **Cosine Similarity**, which measures the angle between two vectors (ranging from -1 for opposite to 1 for identical direction), effectively capturing orientation (semantic relatedness) rather than just vector magnitude [Cao et al., 2024]:

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

The quality and characteristics of the chosen embedding model are crucial, impacting retrieval performance and potentially influencing optimal data preparation strategies like chunking [Yuan et al., 2025].

2.4 Knowledge Base Construction: Challenges and Strategies

The quality, structure, and accessibility of the external knowledge base (KB) are paramount to RAG effectiveness [Lan et al., 2024, Verma et al., 2023]. Designing an optimal KB involves navigating several critical considerations in how information is represented and organized for retrieval.

2.4.1 Chunking: Structuring Documents for Effective Retrieval

Given LLM context window limitations [Liu et al., 2023] and the nature of vector-based semantic search, source documents are typically segmented into smaller units or "chunks" before being indexed [Yuan et al., 2025, Verma et al., 2023]. This chunking process is a crucial step, as the granularity and coherence of chunks significantly influence retrieval precision and the quality of context provided to the generator [Jiang et al., 2024, Zhou et al., N/A, Prem Blog, N/A, Chroma Research, N/A]. Various strategies exist, each with inherent trade-offs:

- **Simple Strategies (Fixed-Size, Recursive):** These methods split text based on length or predefined separators (e.g., paragraphs, sentences). While straightforward to implement, they risk breaking semantic units arbitrarily, potentially fragmenting context or separating closely related information [Prem Blog, N/A].
- **Content-Aware and Semantic Chunking:** These more advanced techniques attempt to identify logical boundaries based on document structure (headings, sections) or semantic shifts in the text, often using embedding similarity [Zhang et al., 2025a, Chroma Research, N/A, Zhang et al., 2025b]. The goal is to create chunks that represent coherent ideas or topics, potentially improving retrieval relevance by providing more focused context.
- **The Monolithic Document Challenge:** An alternative is to avoid explicit chunking and treat entire documents as single units. While preserving the original document structure, this approach faces significant practical limitations. Firstly, many documents exceed the input token limits of standard embedding models, necessitating truncation and risking the loss of potentially vital information [Zhang et al., 2024c]. Secondly, embedding large, potentially multi-topic documents can dilute the resulting vector representation, making it difficult for the retriever to identify precise matches for specific queries [Prem Blog, N/A]. This highlights a fundamental tension between preserving broad context and enabling focused retrieval.

Research continues to explore optimal chunking granularity and the development of adaptive strategies that might tailor chunk size based on content density or query type. Techniques like chunk consolidation or hierarchical retrieval aim to bridge the gap, retrieving based on small chunks but providing larger context to the generator [Verma et al., 2023].

2.4.2 Leveraging Metadata in RAG

Beyond the core text content, associating metadata (e.g., document titles, authors, dates, keywords, structural identifiers) with KB chunks offers a promising avenue for enhancing RAG performance [Yuan et al., 2025, Microsoft Learn, nodate, Zhang et al., 2024a, Acceldata Blog, N/A]. Metadata provides valuable context that can refine retrieval and inform generation. Two primary strategies exist for integrating metadata:

- **Metadata Filtering:**

  The dominant and well-established approach involves storing metadata alongside vector embeddings in the vector database. Retrieval queries can then incorporate filters based on these metadata fields (e.g., filtering by date range or document category) either before (pre-filtering) or after (post-filtering) the vector similarity search [Microsoft Learn, nodate]. Pre-filtering, in particular, can significantly improve efficiency and relevance by narrowing the search space upfront.

- **Embedding Concatenated Metadata and Content:** An alternative, more exploratory hypothesis suggests concatenating textual representations of metadata directly with the chunk's content *before* generating the embedding. The theoretical goal is for the embedding model to implicitly learn a combined semantic representation reflecting both content and metadata context. This approach avoids explicit filtering logic but faces significant uncertainties. Its effectiveness is highly contingent on the specific embedding model's architecture and training data; the model may not be capable of effectively utilizing this concatenated structure, potentially treating the metadata as noise or failing to disentangle its semantic contribution from the primary content [Yuan et al., 2025]. Literature explicitly validating this specific concatenation strategy as a generally effective technique appears limited compared to the established practice of filtering or embedding metadata representations separately.

The choice between these strategies involves trade-offs between explicit, controllable filtering and the less predictable outcomes of implicit, model-dependent semantic encoding via concatenation.

2.5 Information Retrieval Techniques in Modern RAG

The retriever component employs techniques from Information Retrieval (IR) to locate relevant knowledge within the KB, building upon the semantic representations provided by embeddings.

2.5.1 Dense Retrieval Implementation: Vector Stores and Search

Modern RAG systems predominantly rely on dense retrieval using the vector embeddings described in section 2.3. Efficiently searching across potentially millions or billions of these high-dimensional vectors necessitates specialized **Vector Databases** (e.g., FAISS [Johnson et al., 2019], Pinecone, Milvus, ChromaDB). These systems utilize Approximate Nearest Neighbor (ANN) search algorithms to provide fast similarity lookups based on metrics like Cosine Similarity, trading negligible accuracy for significant speed improvements required for real-time applications [Johnson et al., 2019].

2.5.2 Enhancing Retrieval: Multi-Stage Processing and Reranking

While dense retrieval is powerful, its performance can be further enhanced, particularly for complex queries or large KBs where the initial set of retrieved documents might not be optimally ranked. Simple dense retrieval might struggle with precise keyword matches or fail to perfectly rank the most relevant results at the very top. To address this, **Multi-Stage Retrieval** architectures are increasingly common [Yuan et al., 2025, Verma et al., 2023]. These typically involve:

1. An initial, fast **Recall Stage:** Uses dense (or sometimes sparse/hybrid) retrieval to fetch a relatively large set of candidate documents (e.g., top 100) that are potentially relevant.

2. A subsequent **Re-ranking Stage:** Employs a more computationally intensive but accurate model to re-evaluate and re-order this smaller candidate set, aiming to push the most relevant documents to the top positions.

This staged approach balances the need for broad initial search with the requirement for high precision in the final context provided to the generator. The reranking stage often utilizes **Cross-Encoders** models [Reimers and Gurevych, 2019]. Unlike Bi-Encoders (which embed query and document independently),

Cross-Encoders process the query and a candidate document *jointly*. This allows attention mechanisms to model the interaction between query and document terms directly, leading to potentially more nuanced relevance judgments compared to Bi-Encoders [Reimers and Gurevych, 2019]. However, because they require a separate computation for each query-document pair, their use is typically confined to re-ranking the limited candidate set from the first stage (e.g., top 20-100 documents) rather than searching the entire corpus.

2.6 Generation: Synthesizing Faithful and Grounded Responses

The final stage involves the generator LLM synthesizing a response based on the original query and the retrieved context. A key challenge here is ensuring the generated output remains faithful to the provided information and avoids incorporating unsupported claims or hallucinations.

Prompt Engineering plays a critical role in guiding the generator towards faithfulness [Prompting Guide, nodate, Zhang et al., 2025c]. Effective RAG prompts typically provide clear instructions alongside the query and context, explicitly directing the model to base its answer **only** on the retrieved snippets, to cite sources where applicable, and potentially to articulate its reasoning process [Es et al., 2023, Li et al., 2024b]. Techniques like **Chain-of-Thought (CoT)** prompting [Wei et al., 2022], which encourages the model to generate intermediate reasoning steps before the final answer, are adapted in RAG to improve transparency and grounding. By making the reasoning process explicit (e.g., first extracting relevant facts from context, then synthesizing them), CoT aims to reduce logical leaps and enhance the likelihood that the final answer is directly supported by the provided evidence. The broader field of prompt engineering for RAG also explores optimal instruction phrasing, clear context demarcation, and methods for handling uncertainty when retrieved information is insufficient or conflicting. Managing the LLM's finite context window also remains a practical consideration, often requiring strategies like truncation or summarization if the retrieved context is too large [Liu et al., 2023, Bao et al., 2024, Verma et al., 2023].

2.7 The Complexities of Evaluating RAG Systems

Evaluating the end-to-end performance of RAG systems presents significant methodological challenges [Es et al., 2023, Li et al., 2024b, Maxim AI Blog, N/A]. Assessing quality requires considering both the retriever's effectiveness and the

generator's faithfulness and relevance, but obtaining reliable ground truth, particularly for relevance, is often difficult and resource-intensive.

2.7.1 The Ground Truth Scarcity Problem

A primary obstacle in evaluating RAG systems, especially the retrieval component, is the **Ground Truth Problem**, also known as relevance judgment scarcity [Voorhees and Harman, 2001]. For any given query, determining the complete set of truly relevant documents within a large knowledge base requires exhaustive human assessment. This manual annotation process is inherently expensive, time-consuming, and difficult to scale, particularly for specialized domains requiring expert annotators [Voorhees and Harman, 2001, Zhang et al., 2024c]. Furthermore, relevance itself can be subjective and multi-faceted, making consistent annotation challenging even with clear guidelines.

This scarcity of comprehensive, manually-verified relevance labels necessitates pragmatic approaches. Traditional **TREC (Text REtrieval Conference)**-style evaluations often rely on pooling techniques, where relevance is judged only for documents retrieved by a set of participating systems [Voorhees and Harman, 2001]. In the context of RAG development, researchers increasingly turn to alternative strategies:

- **Synthetic Data Generation:** Using LLMs to automatically generate evaluation datasets (e.g., question-context-answer triplets) based on the source documents [Zhang et al., 2024b, Li et al., 2024a]. This offers scalability but requires rigorous quality control to ensure the generated questions are realistic, diverse, and challenging.

- **LLM-Assisted Annotation:** Employing LLMs to provide initial relevance scores or classifications for retrieved documents, which are then reviewed and potentially corrected by human annotators [Zheng et al., 2023]. This human-in-the-loop approach aims to balance the efficiency of LLMs with the accuracy of human judgment, mitigating the bottleneck of purely manual annotation while controlling for potential LLM biases [Zheng et al., 2023, Owens et al., 2024, Verga et al., 2024].

Understanding these challenges and the methodologies developed to address them is crucial for designing credible RAG evaluation frameworks.

2.7.2 A Multi-Faceted Metrics Landscape

Given the complexity and limitations in ground truth, RAG evaluation typically relies on a diverse suite of metrics spanning different aspects of performance. As-

sessing the **retrieval component**, particularly when reliable human relevance judgments are available for a subset of queries, often involves standard IR metrics:

- **Precision@k (P@k):**

  Measures the proportion of relevant documents among the top k retrieved items ($P@k = \frac{|R \cap Ret_k|}{k}$). It reflects the accuracy or purity of the top results presented to the user or the generator.

- **Recall@k (R@k):** Measures the proportion of all known relevant documents that are successfully retrieved within the top k results ($R@k = \frac{|R \cap Ret_k|}{|R|}$). It indicates the retriever's ability to find the relevant information available in the KB.

- **Mean Reciprocal Rank (MRR):** Calculates the average of the reciprocal of the rank at which the *first* relevant document is retrieved ($MRR = \frac{1}{|Q|} \sum_{q \in Q} \frac{1}{\text{rank}_q}$). It is particularly useful when finding the first correct piece of information quickly is important.

- **Normalized Discounted Cumulative Gain (NDCG@k):** Evaluates the overall quality of the ranking up to position k. It assigns higher scores to highly relevant documents placed at earlier ranks, discounting the value of relevant items found lower down ($DCG@k = \sum_{i=1}^{k} \frac{rel_i}{\log_2(i+1)}$; $NDCG@k = \frac{DCG@k}{IDCG@k}$). This provides a nuanced measure of ranking effectiveness, considering both relevance levels and position.

Beyond retrieval, evaluation also considers:

- **Operational Metrics:** Quantify practical aspects like latency and computational cost (e.g., API calls, token usage), essential for assessing deployment feasibility [Li et al., 2024b, Zhang et al., 2024c].

- **LLM-as-Judge Metrics:** Employ powerful LLMs to assess qualitative aspects of the retrieved context (e.g., Context Relevance, Context Sufficiency) and the generated answer (e.g., Answer Faithfulness to context, Answer Relevance to query), often using frameworks like RAGAS [Es et al., 2023]. This approach offers scalability for assessing generation quality without needing reference answers. However, it is crucial to acknowledge their limitations. LLM judges are known to exhibit various biases (e.g., positional bias, verbosity bias, preference for outputs from similar models) and their judgments may lack consistency or perfect alignment with human notions of quality [Zheng et al., 2023, Owens et al., 2024, Verga et al., 2024].

Effective RAG evaluation therefore often requires interpreting results from multiple metric types collectively, acknowledging the imperfections and potential biases inherent in automated or LLM-assisted assessments, particularly when comprehensive ground truth is unavailable.

2.8 Chapter Summary

This chapter has surveyed the background literature essential for understanding Retrieval-Augmented Generation. Beginning with the limitations of standard Large Language Models that motivate the need for external knowledge grounding, we explored the fundamental architecture of RAG systems. Key areas of focus included the critical design choices in Knowledge Base construction, particularly the trade-offs involved in document chunking strategies and metadata integration methods. We examined modern Information Retrieval techniques underpinning RAG, such as dense retrieval via vector embeddings and the increasingly prevalent multi-stage retrieval architectures employing cross-encoder re-ranking for enhanced precision. Furthermore, we discussed prompt engineering techniques aimed at ensuring generational faithfulness and the significant challenges inherent in rigorously evaluating these complex systems, including ground truth scarcity and the nuanced use of LLM-as-judge metrics. This review establishes the context and foundational knowledge upon which the specific methodological approaches and experimental investigations of this project are built.

Chapter 3

Methodology

3.1 Overview and Research Questions

This section outlines the systematic approach undertaken to address the research questions. It details the data acquisition and preprocessing methods, the design rationale for the different Knowledge Base (KB) formats, the architecture of the Retrieval-Augmented Generation (RAG) pipeline, and the evaluation plan designed to compare the effectiveness of each KB implementation. The methodology provides a reproducible framework for evaluating KB structures in RAG systems, grounded in the specific context of the HarmonyOS ArkTS developer documentation.

3.1.1 Data Acquisition: Web Scraping

To construct the KBs, the source content was acquired by scraping the official HarmonyOS ArkTS developer documentation website. The process involved:

3.1.2 Data Source

The primary data source selected for this project is the official HarmonyOS developer documentation, specifically focusing on the ArkTS [1] programming language framework. This choice was driven by several factors resolving the issues faced previously:

- **Relevance:** ArkTS represents a modern, evolving technology, making its documentation a pertinent subject for evaluating information retrieval systems designed to assist developers.

[1] https://developer.harmonyos.com/docs/documentation/doc-guides-V5/arkts-overview-0000001810905997-V5

- **Automated Scraping Feasibility:** The documentation is accessible online without login credentials or mandatory API keys, making systematic, automated scraping technically feasible and reproducible. A review of the site's terms confirmed no explicit restrictions against such automated collection for research purposes.
- **Recency Advantage for RAG Evaluation:** With major updates released in late 2024, the core ArkTS knowledge falls outside the typical training data cut-off for prevalent LLMs (often mid-2023 or earlier). This temporal gap is methodologically advantageous, as it forces the RAG system to rely heavily on the information **retrieved** from the knowledge base rather than its own potentially outdated internal parameters. This setup provides a more rigorous testbed for evaluating the effectiveness of different KB structures on the critical retrieval component.

3.1.3 Intelligent Web Scraping

Acquiring and structuring information from vast, dynamic web sources like technical documentation presents challenges for traditional rule-based scraping techniques, which can be brittle and struggle with website variations. To overcome this, our methodology employed a more adaptive approach using Large Language Models (LLMs). While the use of sophisticated agentic web scraping systems is becoming prevalent in large AI labs for data acquisition, open-source implementations and detailed methodologies remain less explored in academic literature. Our approach, detailed below, utilizes LLMs for intelligent structuring within a hybrid scraping framework.

To acquire and prepare this data systematically, our methodology incorporated automated web scraping and structuring, underpinned by several strategic choices:

- **Systematic Crawling:** We employed a Breadth-First Search (BFS) algorithm to navigate the site structure, starting from the ArkTS overview page, ensuring systematic exploration level by level. To confine the crawl exclusively to relevant sections and prevent deviation into unrelated websites, we implemented strict regular expressions matching the target ArkTS documentation URLs.
- **Hybrid Content Extraction:** To ensure comprehensive content capture across all page structures, we implemented a hybrid extraction strategy. Initially, we employed `BeautifulSoup4`[2] for its speed and efficiency in parsing the static HTML structure. For pages requiring client-side JavaScript execution to render content fully, the system automatically fell back to using

[2] https://www.crummy.com/software/BeautifulSoup/

`Playwright`[3] to load the page dynamically within a headless browser, thereby maximizing content completeness.

- **Agentic LLM Content Structuring:** To adaptively structure extracted content, we employed an agentic LLM workflow, depicted in Figure 3.1 (primarily using DeepSeek models via Groq API). This multi-step process involved initial content preparation, extracting key elements like code and tables, using LLMs for semantic segmentation, metadata generation, and question generation, and incorporating verification stages to ensure accuracy and adherence to a strict JSON schema (see fig. 4.3). This approach provided robust structuring compared to traditional methods. Full implementation details are provided in section 4.3.1.
- **Data Storage:** The structured JSON output from the LLM agent workflow, containing text content chunks and extracted metadata (URL, title, keywords, potential questions, etc.), was stored for subsequent processing by the KB Transformer.

Agentic Content Structuring

Leveraging an LLM (primary: DeepSeek via Groq API; fallback: GPT-4 Turbo), the raw HTML content was processed to extract structured information. This *agentic* approach, while offering flexibility compared to brittle rule-based systems, introduces considerations regarding LLM reliability and computational cost. Illustrated conceptually in Figure 3.1, it involved prompting the LLM to perform semantic chunking, metadata extraction (title, category, keywords, potential questions), code block preservation, and formatting into a predefined JSON schema (detailed in fig. 4.3). The agent was also instructed to identify and flag purely navigational pages. This automated structuring aimed to produce richer, more usable input for subsequent KB transformation stages compared to simple text extraction.

The rationale for employing this agentic, LLM-driven structuring approach, despite the associated computational overhead, stems from the inherent variability and complexity of websites. Automated scraping across numerous pages demanded a flexible method capable of adapting to diverse layouts and content types more effectively than brittle, rule-based systems. Using LLMs allowed for an "intelligent" structuring process that interpreted semantic boundaries and extracted complex elements robustly. This approach aimed to produce consistently structured, high-quality input essential for the subsequent KB transformation stage. Measures taken to ensure the consistency and quality of the LLM-generated structured data

[3] https://playwright.dev/

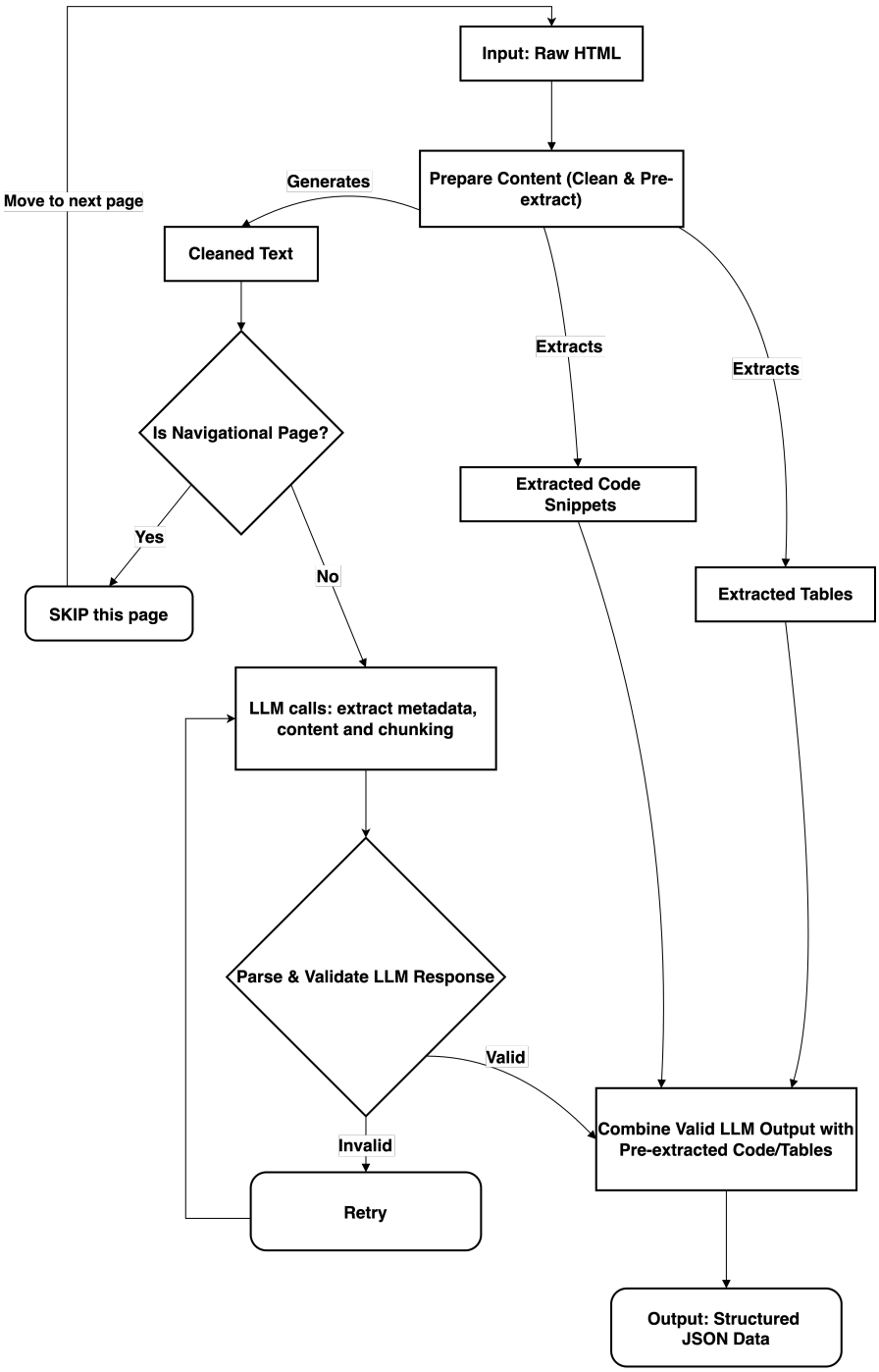


Figure 3.1: Conceptual workflow of the agentic LLM-driven content structuring process.

are discussed further in the evaluation section. Detailed implementation specifics of the scraper are provided in chapter 4, section 4.3.1.

3.2 Knowledge Base Design Rationale

Designing distinct Knowledge Base (KB) formats from this common data source was central to our methodology. Creating multiple formats allowed us to systematically isolate and evaluate the impact of specific structural choices on downstream RAG performance, directly addressing our research questions (RQ1, RQ3). We designed three formats, visually contrasted in Figure 3.2:

- **Monolithic KB Design:** This format represents the simplest approach, treating each original source document (e.g., a scraped webpage) as a single, large entry in the knowledge base. The design rationale was twofold: first, to serve as a baseline representing a naive implementation that preserves the full original context of a document without pre-segmentation; second, to investigate the downsides of retrieving excessively large, undifferentiated blocks of text, which might contain irrelevant information alongside relevant passages. This approach inherently risks significant information loss for longer documents due to embedding model token limits requiring truncation (section 3.2.3).

- **Chunked KB Design:** This format implements a standard and widely adopted strategy in RAG systems. Documents are segmented into smaller, more focused chunks based on the semantic structure identified during scraping (e.g., sections delineated by headings). The design rationale is that retrieving smaller, topically coherent chunks should improve retrieval precision for specific queries compared to monolithic documents, providing more relevant and less noisy context to the generation model. This directly tests the effectiveness of pre-segmentation, although imperfect chunk boundaries remain a potential issue.

- **Enhanced KB Design:** This format builds upon the chunked structure by incorporating additional semantic metadata identified during the scraping process (such as document category, chunk heading, keywords, and relevant questions) directly alongside the chunk's text content. The design hypothesis (RQ3) is that making this metadata explicitly available **at the time of vector embedding** might lead to richer, more contextually aware representations compared to embedding content alone. Rather than employing the well-established technique of metadata filtering during retrieval (section 2.4.2), this project specifically chose to investigate the less explored,

intuitive approach of concatenation. The rationale was to directly test the capability of the chosen embedding model (BGE-base-en-v1.5) to handle and leverage this mixed content input, exploring whether it could implicitly capture the combined semantics without requiring explicit filtering logic, even acknowledging the potential risks of noise or ambiguity discussed in the background.

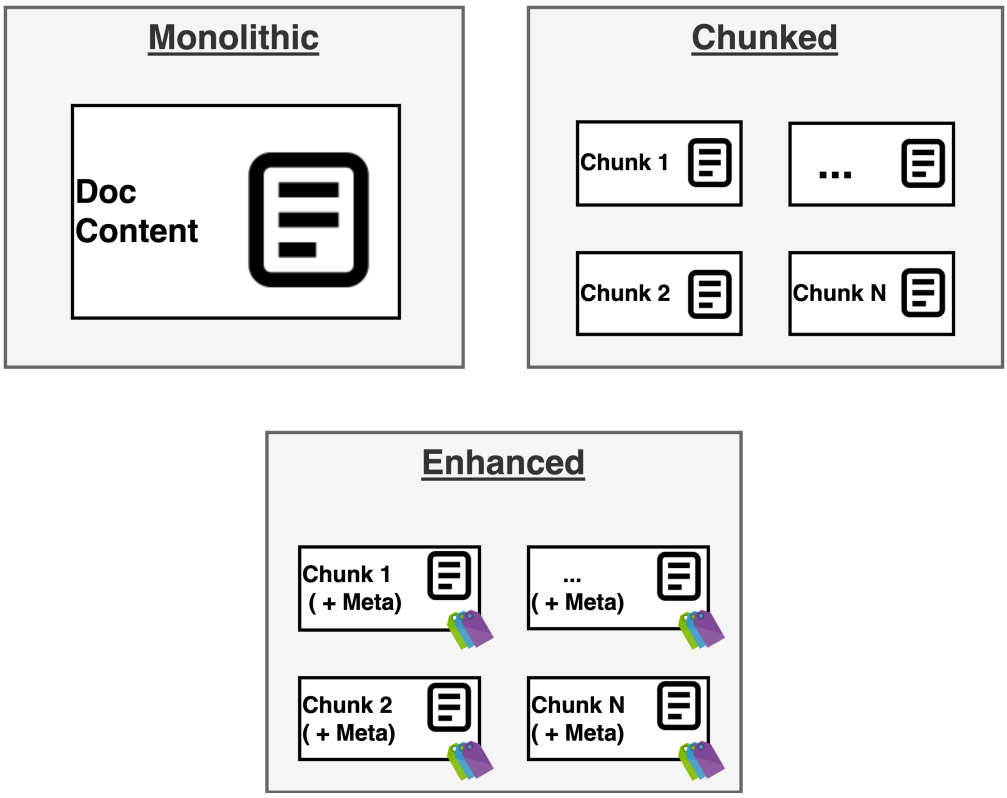


Figure 3.2: Conceptual Comparison of the Three Knowledge Base Structures

The controlled variation across these KB formats, generated via the transformation process detailed in the following section (section 3.2.1), formed the basis of our comparative experimental design.

3.2.1 Knowledge Base Transformation

Following scraping and structuring, the extracted JSON data was transformed into the three distinct KB formats under evaluation (Monolithic, Chunked, Enhanced) as depicted conceptually in Figure 3.2. This transformation step is crucial for isolating the impact of KB structure (RQ1, RQ2) and metadata enrichment (RQ3) on downstream RAG performance.

The transformation of the structured scraper output into the three distinct KB formats (`monolithic`, `chunked`, `enhanced`) was orchestrated by a dedicated KB Transformer component:

- **Orchestration and Modularity:** A central 'KBTransformer' class managed the workflow, invoking specific methods for generating each KB type, ensuring separation of concerns.
- **Incremental Processing:** We implemented an 'UpdateTracker' mechanism utilizing file modification timestamps. This allows the transformer to process only files that are new or updated since the last run, reducing processing time and computational resources used.
- **Parallel Processing:**

  Leveraging Python's 'concurrent.futures.ProcessPoolExecutor', we parallelized the transformation of individual source documents across multiple CPU cores to expedite the overall process.
- **Format Logic:**
  - ***Monolithic:*** The content from all chunks within a source document was merged into a single text block, preserving basic formatting like headings and code blocks.
  - ***Chunked:***

    Each semantic chunk identified by the scraper was processed. Based on a configured threshold, we applied heuristic logic to consolidate consecutive chunks smaller than this threshold. We did this consolidation to avoid overly fragmented retrieval results from very small chunks generated by the scraper, while preserving semantic coherence by merging only adjacent chunks assessed as topically related (based on content and heading similarity). The resulting chunks were then saved as separate files within a document-specific directory.
  - ***Enhanced:*** Similar to the chunked format in structure, but involved significant enrichment **within the transformer**. In addition to using scraper extracted metadata (category, etc.), the transformer applied further heuristic processing: extracting keyphrases from content/headings (via regex), matching questions (from the source document) to specific chunks (via term overlap). This enriched metadata was saved within each chunk's JSON file.

This transformation process ensured that all three KB formats were derived consistently from the same underlying structured data, enabling a controlled comparison. Further implementation details of the KB Transformer can be found in chapter 4.

3.2.2 RAG Pipeline Design

The RAG pipeline was designed with modularity to facilitate the comparison of different KB formats. It follows a standard retrieve-then-read architecture, incorporating a two-stage retrieval process to balance efficiency and effectiveness.

Key considerations guiding the pipeline's design included:

- **Two-Stage Retrieval Strategy:** Recognizing the trade-offs between retrieval speed and accuracy [Verma et al., 2023, Jiang et al., 2024], we employed a two-stage retrieval process. This strategy conceptually involves:
  - An initial, fast candidate retrieval stage using vector similarity to identify a broad set of potentially relevant documents.
  - A subsequent, more computationally intensive but accurate re-ranking stage using a CrossEncoder model to refine the selection and identify the most relevant documents for the final context.

  This approach aims to balance the efficiency needed to search large KBs with the precision required for high-quality generation. The rationale for specific model choices is:
  - *Embedding Model Rationale:* The `BAAI/bge-base-en-v1.5` model was chosen for its strong performance on retrieval tasks (MTEB benchmark), favorable balance between model size and embedding quality, and proven effectiveness in zero-shot RAG scenarios.
  - *Re-ranking Model Rationale:* The `cross-encoder/ms-marco-MiniLM-L-6-v2` model was selected for its high effectiveness (trained on MS MARCO), its status as a standard precision enhancement approach, and its efficient MiniLM architecture offering a good accuracy/speed trade-off.

  This rationale is further underpinned by the need to provide highly pertinent context to the generator. For the `enhanced` KB, this strategy was augmented by embedding specific metadata along with the text content, aiming to improve the semantic signal captured by the initial vector search (RQ3). Full implementation details, including the specific models, parameters, and algorithm, are provided in section 4.3.3 and illustrated conceptually in Figure 4.6.

- **Grounded Prompting Strategy:** The prompt engineering stage explicitly integrates retrieved context with the original query, providing clear instructions to the generation LLM (e.g., use only provided context, cite sources accurately). Furthermore, drawing inspiration from Chain-of-Thought [Wei et al., 2022] and related techniques designed to elicit reasoning, the LLM is instructed to provide not only the final answer but also a separate justification explaining how the answer was derived from the provided context. This explicit request for reasoning aims to improve answer faithfulness and grounding by influencing the LLM's internal generation process towards contextually supported steps, using the autoregressive nature of token prediction. The specific prompt structure and instructions used are detailed in section 4.3.3.

This pipeline design provides the necessary engine for executing the comparative experiments defined by RQ1 and RQ2. The comprehensive framework designed to evaluate the performance implications of these methodological choices across the different Knowledge Base structures is detailed in Chapter 5.

3.2.3 Methodological Considerations and Limitations

It is important to highlight specific limitations stemming from the design choices made within this methodology, distinct from the evaluation process limitations discussed later:

- **Enhanced KB Metadata Utilization Strategy:** The decision to embed metadata **with** the text for the `enhanced` KB tests a specific hypothesis (RQ3) about implicit semantic enrichment via concatenation before embedding. The effectiveness hinges on the chosen embedding model's ability to leverage this combined representation. This contrasts with explicit filtering strategies and means the results specifically reflect this particular metadata integration approach.
- **Monolithic Document Truncation:** The embedding model employed (`BAAI/bge-base-en-v1.5`) possessed a 512-token input limit, a common constraint for high-performance models available during the project's timeframe. This necessitated truncating ∼67% of monolithic documents, a deliberate choice made to directly expose the practical limitations of a naive monolithic approach when facing such contemporary hardware/model constraints, rather than applying a potentially confounding chunking strategy to this baseline KB. While the trend is towards models with increasingly large context windows and increasing input limits, the sheer volume of technical

documentation encountered in practice and the associated computational costs of processing vast contexts mean that efficient indexing and retrieval strategies, such as chunking, retain significant practical relevance irrespective of theoretical maximum context sizes.

These factors should be considered when interpreting the comparative results presented in Chapter 5.

Chapter 4

Implementation Details

4.1 Technology Stack

This project was implemented primarily in Python (developed and tested using Python 3.11), leveraging several key libraries and tools detailed in Table 4.1.

4.2 System Architecture Diagram

The overall system architecture, conceptually illustrated in fig. 4.1. This chapter details the specific implementation choices for the core components outlined conceptually: the Scraper, KB Transformer, RAG Pipeline, and Evaluation Framework.

4.3 Module Implementation Details

This section provides specific implementation and configuration details for the core pipeline outlined in the methodology (chapter 3).

4.3.1 Scraper Implementation

The scraping process involved several stages:

Crawling

Web navigation utilized a Breadth-First Search (BFS) approach. The crawl scope was controlled by configurable parameters including a maximum depth (`max_depth=4`) and a maximum number of pages (`max_pages=100`) per crawl. This was used to break up the crawl into manageable chunks, and to ensure the

Table 4.1: Core Technology Stack

Category	**Libraries/Tools**
Web Scraping & HTML Processing	`requests`, `BeautifulSoup4`, `Playwright`, `aiohttp`
LLM Interaction	`openai` Python client, direct API calls via `aiohttp` (for Groq endpoint)
Vector Store & Embeddings	`chromadb-client`, `sentence-transformers` (specifically `BAAI/bge-base-en-v1.5`)
Core RAG/NLP	`transformers` (Hugging Face)
Data Handling & Transformation	`concurrent.futures`, `json`, `re`, `logging`, `asyncio`, `Pandas`, `NumPy`
Evaluation Metrics	`nltk`
Visualization	`matplotlib`

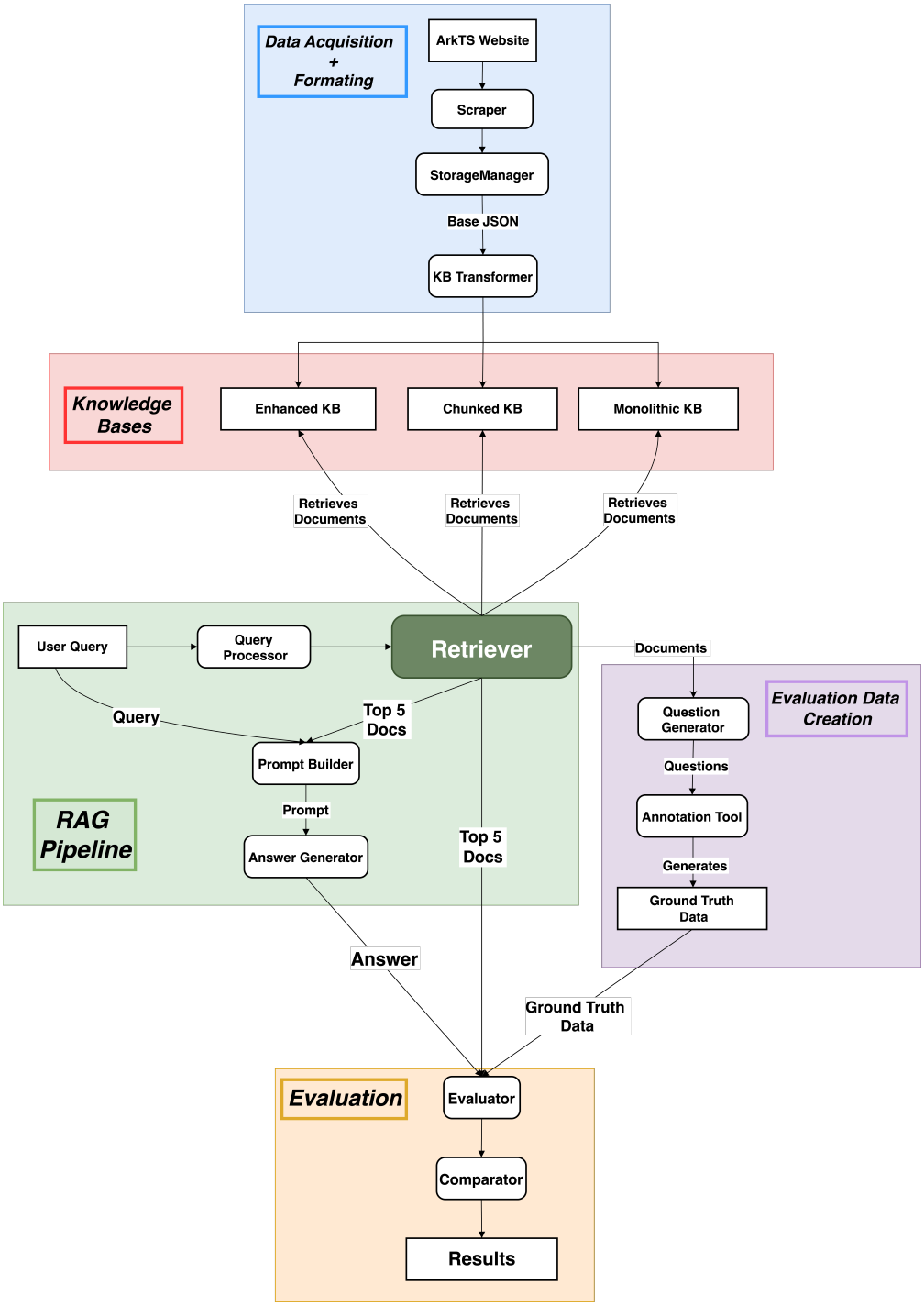


Figure 4.1: High-level architecture of the RAG system and evaluation workflow.

crawler did not get stuck in an infinite loop. Navigation was restricted to the target ArkTS documentation URLs using regular expressions to ensure domain relevance. URL tracking (visited, processed, errors) was handled internally by a dedicated storage management component.

Hybrid Content Extraction

A hybrid strategy balanced speed and completeness. Initially, asynchronous HTTP requests (`aiohttp`) combined with `BeautifulSoup` parsing were used for efficient static HTML extraction. If internal heuristics (e.g., based on content length or structure analysis) determined the extracted content might be incomplete (common with JavaScript-heavy pages), the system automatically triggered a fallback mechanism using `Playwright`'s asynchronous API to render the page fully in a headless browser, ensuring dynamic content was captured.

LLM-Driven Content Structuring and Configuration

The agentic LLM workflow described in the methodology was implemented using specific models and prompt strategies. The primary configuration prioritized a DeepSeek model accessed via the Groq API. If this was unavailable, the system defaulted to `gpt-3.5-turbo` accessed via the standard `openai` Python client.
The core logic relied on a detailed system prompt instructing the configured LLM to transform the raw HTML input into a structured JSON object suitable for RAG. The prompt emphasized the need for completeness (D̈ONT MISS !ANY! DETAILS̈), explicitly required top-level fields (`doc_id`, `title`, `last_updated`, `chunks`, `potential_questions`, `metadata`), detailed the required structure within the `chunks` array (`heading`, `content`, optional `code_snippets`, chunk `metadata`), specified how to handle tables (as arrays of objects), and included a rule to output `?SKIP?` for purely navigational pages. Output was constrained to valid JSON strictly enclosed within `BEGIN_JSON` and `END_JSON` markers. Robust standard Python `json` parsing with try-except blocks and automated retry mechanisms were implemented downstream to handle potential LLM API errors or malformed JSON outputs.

4.3.2 KB Transformer Implementation

The KB transformation component converted the structured JSON outputs from the scraper into the three distinct KB formats being evaluated, as illustrated in fig. 4.4. Key implementation aspects included:

- Keyphrase extraction using regular expressions targeting code-like identifiers (PascalCase, camelCase, snake_case), quoted terms (3+ characters), and

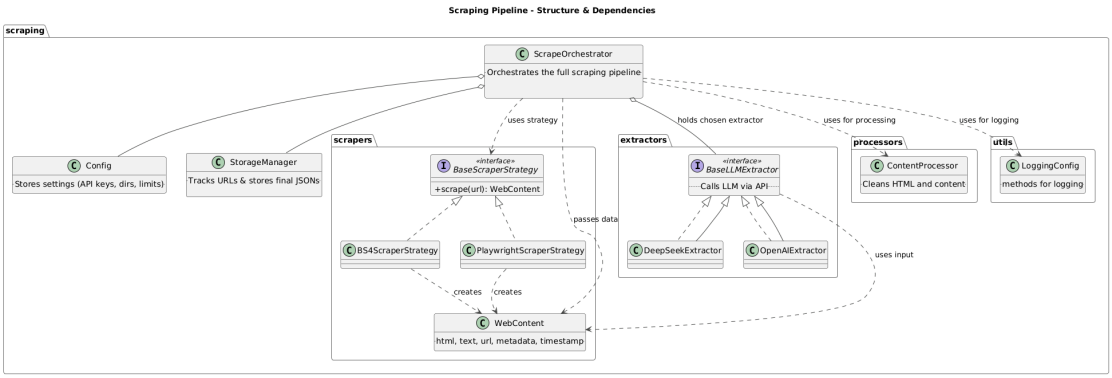


Figure 4.2: UML diagram illustrating the scraping pipeline architecture.

simplified capitalized noun phrases (e.g., Ínitial Value; filtered for length > 5).

- **Efficiency:**

  Implemented incremental updates by tracking file modification timestamps (`os.path.getmtime`) stored in a tracking file, processing only new or modified source JSONs. Utilized `concurrent.futures.ProcessPoolExecutor` to parallelize the transformation of individual documents across multiple CPU cores.

The specific transformation logic for each format was implemented as follows:

- **Monolithic Format:**

  Concatenated the `content` field from all chunks within a source JSON into a single text block. Basic structure was preserved by prepending headings with Markdown syntax (e.g., `##`) and attempting to render code/tables intelligibly. Output was saved to a dedicated monolithic KB directory. The handling of documents potentially exceeding downstream embedding model limits is addressed during the embedding process (section 4.3.3).

- **Chunked Format:**

  Processed individual semantic chunks identified by the scraper. Applied a consolidation heuristic: consecutive chunks were merged if their combined character count was below a threshold (`consolidate_threshold=200`) and they were deemed semantically related. Semantic relatedness was determined first by checking for direct term overlap (intersection) between the sets of non-stopword tokens in the headings, and secondarily by checking if any pair of significant heading terms matched predefined contextual pairs (e.g., (över-view; ïntroduction)). The resulting potentially consolidated chunks were

```
"url": "< URL of the scraped page>",
"title": "<Document Title>",
"last_updated": "<YYYY-MM-DD>",
"chunks": [
  {
    "heading": "<Section heading or title>",
    "content": "<Main content text...>",
    "metadata": {
        "category": "<e.g., App Development>",
        "importance": "<high|medium|low>",
        "audience": ["developers"]
    },
    "code_snippets": [
        {
            "description": "<Desc...>",
            "code": "<Code snippet>"
        }
    ]
  }
],
"potential_questions": [
  "<Question 1>",
  "<Question 2>"
],
"metadata": {
  "source": "<playwright|bs4>",
  "priority_level": "<important|informational>",
  "keywords": ["ArkTS", "..."],
  "source_url": "<URL>"
}
```

Figure 4.3: Example JSON Schema for Scraper Output

saved as separate JSON files within a document-specific subdirectory in the chunked KB path, along with an index file listing the constituent chunk IDs.

- **Enhanced Format:** Followed the same directory and chunk file structure as the Chunked format but included significant heuristic enrichment applied by the transformer before saving each chunk JSON. This added metadata fields derived via:
  - Potential question matching by tokenizing the question and chunk (heading+content), calculating normalized term overlap, boosting the score by +0.3 if any heading term appeared in the question, and assigning relevance if the final score exceeded 0.6 OR the raw overlap count exceeded 5 OR (a heading term matched AND the raw overlap count exceeded 3).

  The index file for the document was also enriched with document-level metadata. As detailed later (section 4.3.3), this enriched metadata was utilized during the RAG pipeline's embedding stage via pre-concatenation to test RQ3.

4.3.3 RAG Pipeline Components Implementation

The core RAG workflow, realizing the design from section 3.2.2, was implemented with the following component configurations:

Retriever

The retrieval subsystem implemented the two-stage process outlined in the methodology (section 3.2.2) using these specifics:

- **Vector Store:** Utilized the `chromadb-client` library to manage persistent ChromaDB collections stored on disk. Separate collections were maintained for each KB format (`monolithic`, `chunked`, `enhanced`).
- **Embedding Model and Process:** Used the `sentence-transformers` library to load the `BAAI/bge-base-en-v1.5` model. An adapter class integrated this model with ChromaDB's embedding function. For the `monolithic` KB, documents exceeding the model's 512-token input limit were truncated during this embedding process, consistent with the naive strategy limitation discussed in section 4.4.1. This adapter handled embedding both the user query and the candidate document texts.

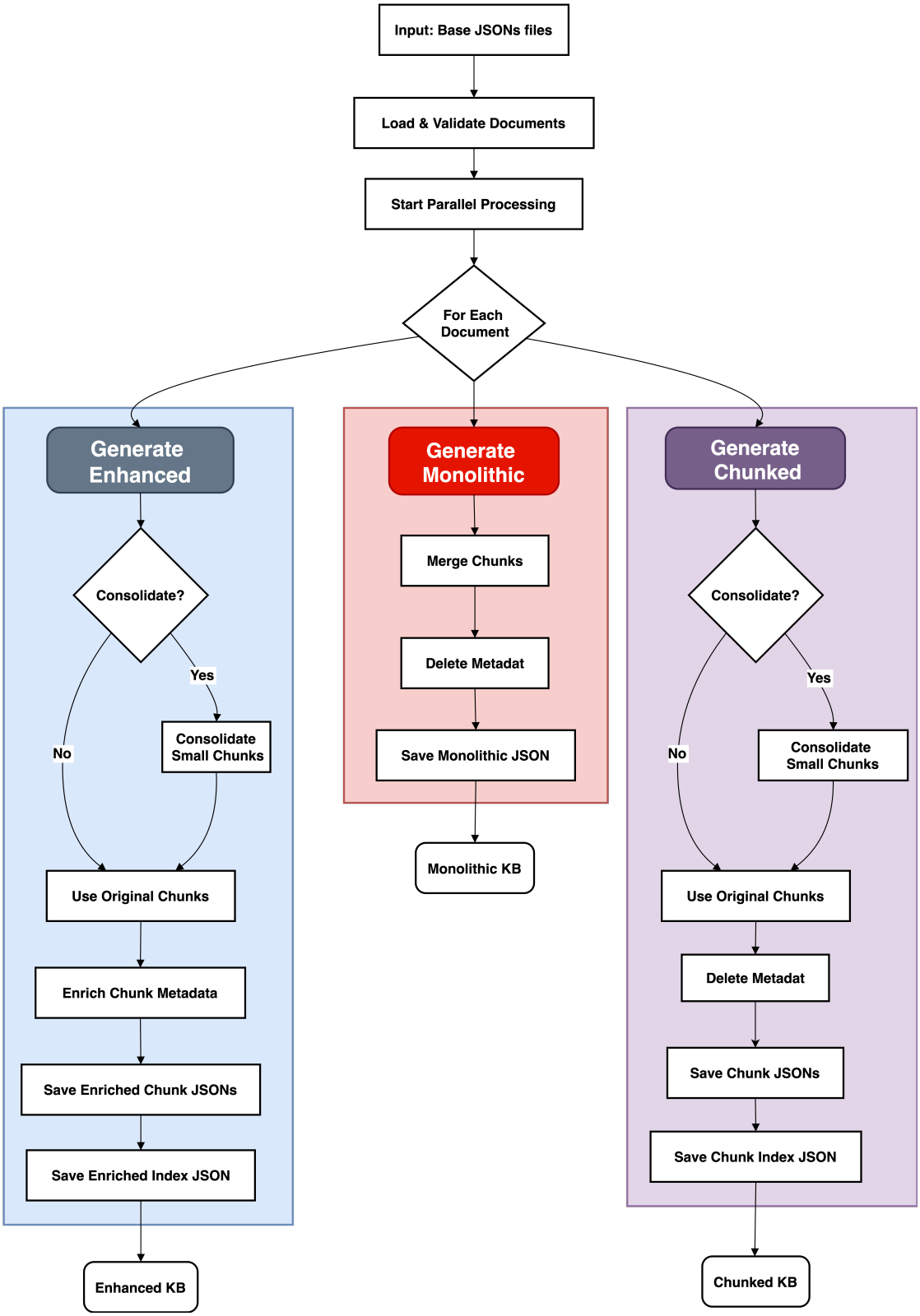


Figure 4.4: Flowchart illustrating the KB transformation process generating Monolithic, Chunked, and Enhanced formats.

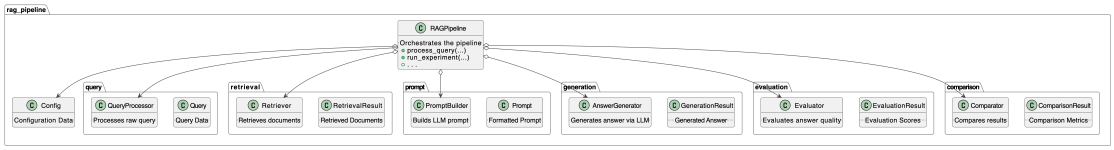


Figure 4.5: UML component diagram of the RAG Pipeline.

- **Embedding Input Strategy:** The text prepared for the embedding model varied by KB type. For the **Enhanced KB**, key metadata fields (Category, Heading, scraper-generated Keywords, scraper-generated Questions) were concatenated with the chunk's content before embedding, aiming to create a richer semantic representation (RQ3). For **Monolithic** and **Chunked KBs**, only the primary text content was embedded. Metadata intended for ChromaDB storage was appropriately sanitized, and the concatenation for the enhanced KB was performed using natural language phrasing to maintain compatibility with the embedding model.

- **Two-Stage Retrieval Execution:**

  The retrieval process unfolded in two stages, as illustrated in Figure 4.6. First, an **initial vector search** queried the appropriate ChromaDB collection using the embedded user query (cosine similarity) via the `chromadb-client`, retrieving an initial set of `n_results = 20` candidates (parameter `rerank_top_n`). Second, if re-ranking was enabled (`use_reranker=True`), these candidates underwent **Cross-Encoder re-ranking**. This used a `CrossEncoder('cross-encoder/ms-marco-MiniLM-L-6-v2')` model loaded via `sentence-transformers` to compute relevance scores for each `(query, candidate_document_text)` pair. Finally, a **selection** step sorted the candidates by the CrossEncoder score (or by the initial vector similarity score if re-ranking was disabled) and selected the top `top_k = 5` documents to form the final context. This set of documents, along with their metadata and scores, was passed to the Prompt Builder.

Prompt Builder

The prompt building component constructed the final prompt sent to the generator LLM, implementing the Grounded Prompting Strategy described in section 3.2.2. Its responsibilities included:

- **Context Assembly and Formatting:** It received the sorted list of `top_k=5` retrieved documents from the Retriever. The context string was assembled by iteratively adding formatted entries for each document, ensuring the final

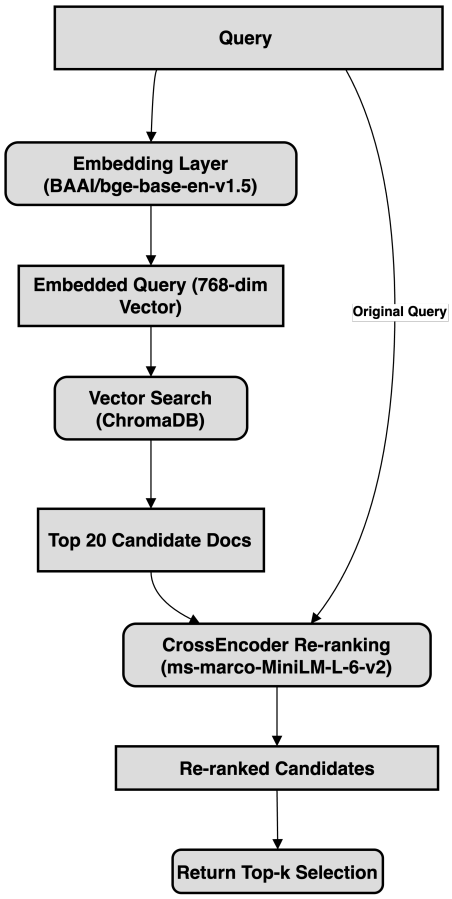


Figure 4.6: Flowchart illustrating the two-stage retriever process implementation.

context adhered to configured limits for the number of entries (`MAX_CONTEXT_ENTRIES=5`) and total characters (`MAX_CHARS=3500`). Individual document entries exceeding a character limit (`MAX_ENTRY_LENGTH=1200`) were truncated.

- **Prompt Engineering and Structure:** The component integrated the assembled context string with the user's original query into a structured prompt designed to guide the generator LLM. Following the principles outlined in the methodology, the prompt included specific instructions reinforcing faithfulness to the provided context and source attribution. To illustrate the specific instructions and formatting, the core prompt template structure is shown in Figure 4.7.

```
System Prompt:
You are an AI assistant specializing in HarmonyOS ArkTS development.
Answer the user's query based \textbf{only} on the provided context documents.
Do not use any prior knowledge.
Cite the source document ID (e.g., [Source doc_id]) for information used in your an
Provide a concise answer to the query.
Then, provide a separate justification explaining step-by-step how you derived the

Context:
[Formatted context entries based on Algorithm 2 go here...]
[Source doc_1_id, Score: 0.95]
Content of document 1...

[Source doc_2_id, Score: 0.92]
Content of document 2...

User Query:
[User's original question goes here...]

Answer:
[LLM generates answer here...]

Justification:
[LLM generates justification here...]
```

Figure 4.7: Simplified structure of the prompt template used for the generator LLM.

Answer Generator

Handled interaction with the generation LLM APIs:

- **LLM Configuration:** Primarily configured to use the OpenAI API via the `openai` client targeting `gpt-4-turbo-preview`. Included a fallback mechanism to the Groq API endpoint (targeting `llama370b-8192`) if the primary OpenAI call failed.
- **Parameters:** Central configuration settings defined default generation parameters: `temperature=0.1` (for determinism).
- **Output:** Returned the generated text along with token usage statistics (extracted from the API response `usage` field), generation time, and the provider used.

The implementation details for the components responsible for executing the evaluation plan are described in Chapter 5.

4.4 Testing Strategy

To ascertain the correctness and robustness of the RAG pipeline and its constituent modules, a rigorous testing methodology was employed, leveraging the `pytest` framework and Python's `unittest` library. This strategy adopted a multi-layered approach, encompassing distinct levels of validation to ensure comprehensive coverage:

- **Unit Testing:** A suite of unit tests was developed using `pytest` to verify the functional correctness of individual components in isolation. These tests utilized mocking techniques (via `unittest.mock`) to isolate dependencies and provide deterministic inputs. Key functionalities subjected to unit testing included the calculation logic for various evaluation metrics within the `Evaluator` (e.g., verifying NDCG calculations against known relevant/retrieved sets), the mechanisms for loading different Knowledge Base formats within the `Retriever`, and the prompt construction logic within the `PromptBuilder` across different KB configurations. Specific helper classes, such as those responsible for text similarity computations, were also rigorously tested against predefined inputs and expected outputs.
- **Integration Testing:** To validate the interactions and data flow between collaborating components, integration tests were implemented. These tests verified critical sequences within the RAG pipeline, such as the flow from

KB loading through retrieval, the integration of retrieval results into prompt construction, the handover from prompt building to answer generation, and the subsequent processing of generation results by the evaluation framework. Successful execution and correct data marshalling between stages were the primary validation points.

- **End-to-End Testing:** Full pipeline tests were designed to simulate the complete request lifecycle, from initial query processing to final evaluation output. These tests executed the primary pipeline entry points, processing queries against the different KB configurations (monolithic, chunked, enhanced). The objective was to confirm the seamless orchestration of all components and validate the structure and content of the final results. It should be noted that certain end-to-end tests necessitated external API interactions for embedding and generation services.

- **LLM-as-Judge Meta-Evaluation:** Recognizing the reliance on LLMs for certain crucial tasks (e.g., generating relevance scores during annotation, potential future data extraction), a supplementary evaluation layer was introduced. This involved employing a secondary LLM (GPT-4) guided by carefully designed prompts. This 'LLM-as-judge' assessed the quality, faithfulness, and coherence of outputs generated by the primary LLMs within the system. This provided an automated mechanism for quality control, specifically targeting the mitigation of potential issues like factual inaccuracies (hallucinations) or poorly reasoned justifications in LLM-generated content, thereby enhancing confidence in automated processes.

This layered testing strategy aimed to provide robust validation, covering individual component logic, inter-component communication, and the overall system workflow. The LLM-as-judge component added a further quality assurance dimension specific to the AI-driven aspects of the pipeline.

4.4.1 Limitations

A key limitation inherent in the chosen implementation, particularly affecting the **Monolithic KB**, is the handling of documents exceeding the input token limit (512 tokens) of the `BAAI/bge-base-en-v1.5` embedding model. As outlined in section 4.3.3, the strategy employed simple truncation. This naive approach can lead to significant information loss, potentially discarding relevant content from the latter parts of long documents before they are embedded. This truncation directly impacts the quality of the vector representation for such documents and consequently hinders retrieval effectiveness for queries whose answers lie in the

truncated sections. While necessary for compatibility with the model, this represents a known drawback of the monolithic approach compared to chunking strategies which inherently manage document length.

Chapter 5

Evaluation Framework and Results

5.1 Introduction

Addressing the research questions outlined in section 1.4 necessitates a rigorous evaluation of the implemented RAG pipeline across the different Knowledge Base configurations. Evaluating complex AI systems like RAG presents significant challenges, particularly the "Ground Truth Scarcity Problem" when dealing with large, domain-specific corpora without pre-existing relevance labels. Determining the definitively correct set of documents for any given query across thousands of potential candidates requires extensive human effort, often infeasible within project constraints.

Therefore, a multi-faceted evaluation strategy, combining objective ground-truth metrics on a subset of data with scalable LLM-based assessments and operational benchmarks, was essential for drawing credible and nuanced conclusions. This chapter details this comprehensive evaluation approach and presents the empirical findings. It begins by outlining the *methodology* employed for creating evaluation data and defining metrics (section 5.2). It then describes the technical *implementation* of the evaluation framework components (section 5.3). Next, it characterizes the *evaluation dataset* used (section 5.4). Following this, the *experimental setup* is detailed (section 5.5). The core *results* are then presented, covering ground-truth retrieval performance, LLM-evaluated quality metrics, and operational benchmarks (section 5.6). Subsequently, these findings are synthesized and discussed in relation to the research questions and hypotheses (section 5.7). Finally, the chapter acknowledges the inherent *limitations* of the evaluation process itself (section 5.8).

5.2 Evaluation Methodology

A rigorous evaluation plan underpins the comparative analysis presented in this chapter. This section details the methodology for constructing the evaluation dataset and the metrics used to assess performance across multiple dimensions, thereby establishing the framework for quantitatively addressing the research questions.

5.2.1 Evaluation Question Generation

A crucial aspect of evaluating RAG systems is the quality and relevance of the test questions. Generating these questions manually is time-consuming and may introduce bias. To address this, an automated approach was developed, leveraging LLMs to generate evaluation question/answer pairs grounded in the source documentation. This methodology ensures that the evaluation directly reflects the KB's ability to answer questions derived from its own content. The process involved:

- **LLM-Based Generation:** Utilising LLMs (DeepSeek models) prompted with content from the source documentation to generate initial question drafts.
- **Iterative Refinement:** Implementing successive experimental refinements focusing on document organization (topic pooling, usage tracking), structured prompting (explicit complexity, multi-document reasoning), sophisticated deduplication (Jaccard similarity), balancing single vs. multi-document questions via specialized pipelines, and tuning the final complexity distribution.
- **Filtering and Quality Control:** Applying automated filters for length and format, alongside intelligent deduplication logic, to ensure the final dataset's quality and diversity.

This iterative process yielded the final evaluation dataset described in section 5.4.

5.2.2 Ground Truth Creation Methodology

Addressing the "Relevance Judgment Scarcity Problem" [Voorhees and Harman, 2001] for our large corpus required a dedicated methodology for creating a reliable ground truth dataset for a subset of questions. We developed and utilized a semi-automated annotation tool implementing the workflow illustrated in Figure 5.1, based on the following principles:

- **Semi-Automated Workflow:** LLMs (DeepSeek and OpenAI models) were used to generate initial relevance score suggestions (score, explanation) for retrieved documents, accelerating the process.
- **Human Validation:** These LLM suggestions served as intermediate input for a subsequent manual annotation phase. Human annotators reviewed the question, the document content, and the LLM explanation to provide the definitive relevance judgment (on a continuous scale), forming the final ground truth.
- **Retriever Alignment:** To ensure relevance judgments were applicable to the system under test, the annotation tool's document retrieval stage utilized the *exact same core RAG pipeline retriever component* used in the main experiments, including the same embedding models and vector store configurations.
- **Expanded Retrieval (k=30):** During annotation retrieval, a larger `top_k` value of 30 was used (compared to the typical pipeline default of 5). This captured a wider pool of potentially relevant documents, including lower-ranked ones, enabling more accurate calculation of recall-oriented metrics against the human-validated ground truth.

This two-stage, human-in-the-loop approach aimed to balance the need for scale with the imperative for accuracy, generating a high-fidelity, human-validated ground truth dataset for the subset detailed in section 5.4.

5.2.3 Evaluation metrics

Recognising that RAG-system performance is multi-dimensional, we evaluate each knowledge-base configuration with the metric suite summarised in Table 5.1. A mix of ground-truth IR measures, LLM-judged scores and operational figures is necessary, because no single view captures the trade-offs between retrieval effectiveness, answer quality and runtime overhead.

Ground-truth retrieval metrics are computed on the human-annotated subset. Let R be the set of relevant documents for a query and Ret_k the top-k retrieved documents. For a query set Q:

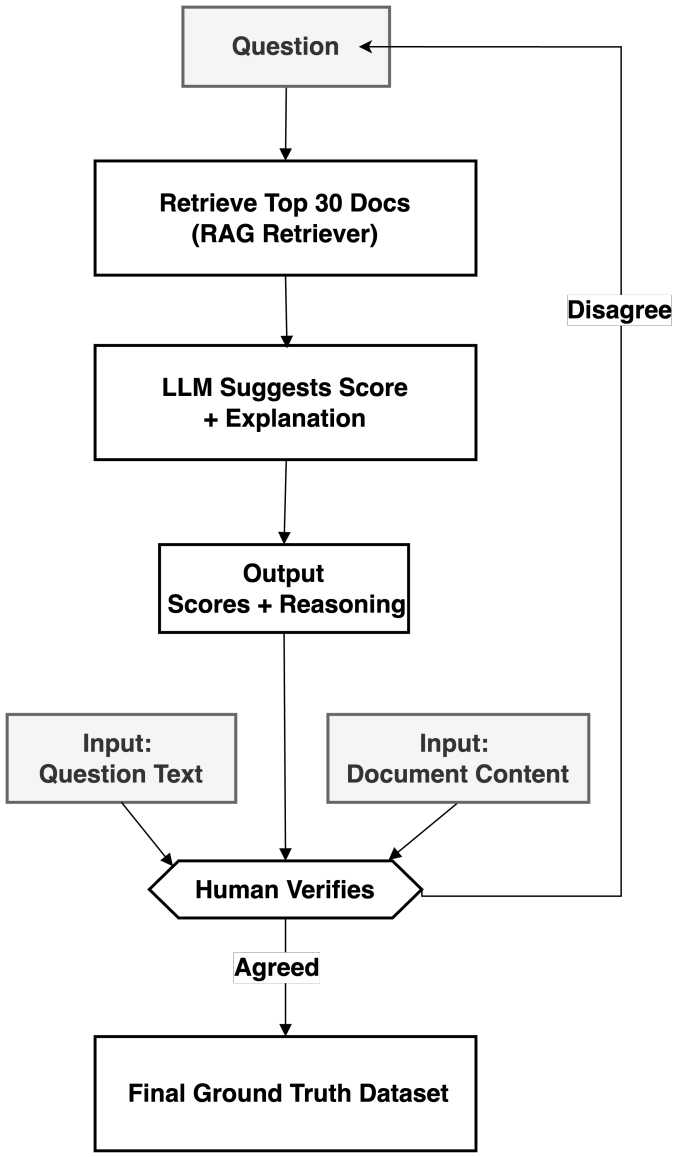


Figure 5.1: Workflow for Semi-Automated Ground Truth Annotation

Table 5.1: Evaluation-metric suite used in this study.

Category	**Metric(s)**
Operational	Latency (total, retrieval, generation); total token cost
Retrieval (ground truth)	Precision@k, Recall@k, mean reciprocal rank (MRR), NDCG@k
Context quality (LLM-eval)	Context relevance, context sufficiency, context pertinence
Answer quality (LLM-eval)	Answer relevance (to query), answer faithfulness (to context), answer completeness (from context)

$$\text{Precision@}k = \frac{|R \cap \text{Ret}_k|}{k}, \tag{5.1}$$

$$\text{Recall@}k = \frac{|R \cap \text{Ret}_k|}{|R|}, \tag{5.2}$$

$$\text{MRR} = \frac{1}{|Q|} \sum_{q \in Q} \frac{1}{\text{rank}_q}, \tag{5.3}$$

$$\text{DCG@}k = \sum_{i=1}^{k} \frac{rel_i}{\log_2(i+1)}, \qquad \text{NDCG@}k = \frac{\text{DCG@}k}{\text{IDCG@}k}. \tag{5.4}$$

The role of each metric category is:

Operational Quantifies latency and cost trade-offs (addresses RQ2).

Retrieval Standard IR measures against human judgements.

Context LLM-judged relevance, sufficiency and pertinence of the retrieved context.

Answer LLM-judged relevance, faithfulness and completeness of the generated answer.

5.2.4 Experimental Hypotheses

Based on RAG principles and our KB designs, we formulated the following hypotheses prior to experimentation:

- H1: The *chunked* KB structure will yield significantly better ground-truth retrieval performance (P@k, R@k, MRR, NDCG) than the *monolithic* structure.

- H2: The *chunked* KB's more focused context will lead to generated answers with higher LLM-evaluated quality (Relevance, Faithfulness, Completeness) compared to the *monolithic* KB.
- H3: The *enhanced* KB, with embedded metadata, will outperform the standard *chunked* KB on retrieval metrics.

Having established the methodological foundation for evaluation, the subsequent section details the specific software components developed to implement this plan.

5.3 Evaluation Framework Implementation

To execute the multifaceted evaluation methodology outlined above, a dedicated software framework was implemented in Python, integrating with the core RAG pipeline and leveraging the technology stack detailed in section 4.1. This framework comprises several key components designed to automate metric calculation and results aggregation:

- **GroundTruth Handler:** Loads the pre-annotated ground truth data (mapping evaluation questions to relevant document IDs and scores) derived from the process in section 5.2.2. Facilitates the calculation of ground-truth retrieval metrics.
- **Comparator:** Aggregates individual evaluation results per query-KB pair. Groups results by KB type and calculates summary statistics (mean, median, std dev) for each metric using libraries like `NumPy` and `Pandas`. Stores results and generates comparative reports/visualizations (using `matplotlib`).
- **Evaluator Core Logic:** Orchestrates the calculation of the full suite of metrics (table 5.1) for each RAG pipeline output. This involves calculating operational metrics, invoking LLM-as-judge assessments for context/answer quality (using specific prompts), and coordinating with the GroundTruth Handler for retrieval metrics.

This implemented framework provides the necessary tooling to conduct the comparative experiments described in the setup section that follows the dataset characterization.

5.4 Evaluation Dataset Characteristics

This section details the evaluation question dataset employed in this study, generated according to the methodology described in section 5.2.1. Understanding the

dataset's composition is essential for interpreting the performance analyses that follow.

5.4.1 Dataset Composition

The final evaluation dataset comprises **1,145** questions derived from the source documentation using the iterative LLM-based process. Questions span a range of complexities relevant to ArkTS development and were categorized based on estimated difficulty (Simple, Moderate, Complex) and the number of source documents typically required for a comprehensive answer (Single-Document, Multi-Document). Table 5.2 presents the distribution. A subset of **111** questions was further subjected to human annotation to establish ground truth relevance judgments (section 5.2.2); this subset forms the basis for the ground truth retrieval evaluation presented in section 5.6.1. The full dataset of 1,145 questions was utilized for the LLM-based evaluations discussed in section 5.6.2 and operational analysis in section 5.6.3.

Table 5.2: Distribution of evaluation-question categories.

Category	**Sub-category**	Count
Complexity	Simple	303
	Moderate	525
	Complex	317
Doc. need	Single-Doc	538
	Multi-Doc	607
Total questions		1145
Ground-truth subset		111
LLM-eval set		1145

Example questions illustrating the different types include:

Example 1 (Simple, Multi-Doc, GT)

"Explain why Vector is deprecated in ArkTS and suggest an alternative for a HarmonyOS app that requires dynamic array functionality."

Example 2 (Moderate, Multi-Doc)

"Explain how to implement application internationalization in ArkTS and describe the role of the LayerDrawableDescriptor in handling cultural

formatting of UI layers."

Example 3 (Complex, Multi-Doc)

"Design a secure file sharing mechanism between two ArkTS applications using the Distributed File System (DFS) and HUKS for key management. Detail the key exchange process, file encryption/decryption steps, and necessary permissions."

5.4.2 Purpose of Question Types

The inclusion of diverse question types serves specific evaluation purposes. Simple, single-document questions primarily test the system's ability to retrieve and synthesize basic factual information accurately. Moderate, complex, and multi-document questions, conversely, challenge the RAG pipeline's capacity for deeper reasoning, context integration across multiple retrieved chunks, and overall robustness when handling more nuanced information needs. Evaluating performance across this spectrum provides a more holistic understanding of each KB structure's strengths and weaknesses.

5.5 Experimental Setup

The comparative evaluation was conducted by executing the RAG pipeline (implementation detailed in section 4.3.3) against the evaluation questions characterized above (section 5.4). Each question was processed sequentially for each of the three Knowledge Base configurations: `monolithic`, `chunked`, and `enhanced`. The pipeline utilized the consistent configurations detailed in Chapter 4 (e.g., BGE embeddings, ChromaDB, CrossEncoder re-ranking, GPT-4-Turbo/DeeSeek generation).

For each query-KB pair, the `Evaluator Core Logic` (section 5.3) calculated the full suite of metrics. These results were systematically logged and aggregated by the `Comparator` component to facilitate the comparative analysis presented next.

5.6 Results

This section presents the empirical findings from the comparative experiments, systematically addressing the hypotheses and research questions through quantit-

ative analysis across the defined metric categories, starting with the ground-truth retrieval assessment.

5.6.1 Ground Truth Retrieval Performance

We first assess retrieval effectiveness against the 111-question subset featuring human-validated relevance judgments (section 5.2.2), using a relevance threshold of 0.4 for binary metrics.

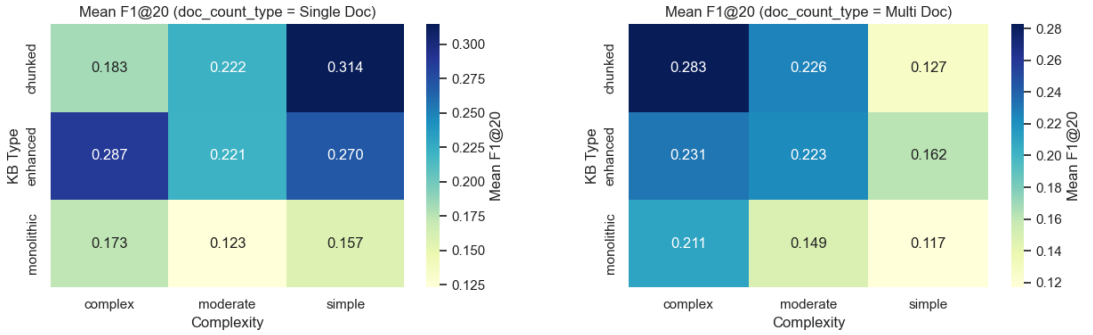
Table 5.3 summarizes the core retrieval metrics at K=10 and K=20. These aggregated results provide strong empirical support for hypothesis **H1**. Both the `Chunked` and `Enhanced` KBs demonstrate markedly superior performance over the `Monolithic` baseline across all standard IR evaluation measures, including key indicators like NDCG@10 (0.585 and 0.621 vs. 0.516) and Recall@20 (0.307 and 0.332 vs. 0.201). Notably, the relatively high Mean Reciprocal Rank (MRR) scores across all configurations, particularly the structured KBs, suggest that the two-stage retrieval process, including the CrossEncoder re-ranking, is effective at bringing at least one relevant document near the top of the list when such documents are initially retrieved. This significant performance differential underscores the critical advantage conferred by semantic chunking in mitigating the negative impacts of document truncation (section 3.2.3) and enabling more focused retrieval compared to processing large, undifferentiated text blocks.

Table 5.3: Ground Truth Metric (Relevance Threshold = 0.4)

	k = 10			**k = 20**		
Metric	Monolithic	Chunked	Enhanced	Monolithic	Chunked	Enhanced
Precision	0.232	0.307	**0.337**	0.158	0.233	**0.241**
Recall	0.157	0.222	**0.255**	0.201	0.307	**0.332**
MRR	0.644	0.641	**0.649**	0.645	0.643	**0.652**
NDCG	0.516	0.585	**0.621**	0.490	0.586	**0.640**

While chunking clearly outperforms the monolithic approach, the F1@20 scores, broken down by question complexity and document requirement in Figure 5.2, reveal distinct performance profiles for the `Chunked` and `Enhanced` structures. Consistent with H1, the `Monolithic` KB yields the lowest F1 scores in most conditions. However, comparing the structured KBs, the `Enhanced` KB exhibits the highest F1 score for complex and moderate single-document queries, whereas the standard `Chunked` KB demonstrates superior F1 performance for complex and moderate multi-document queries. This pattern suggests that while metadata embedding might enhance focus for single-topic retrieval, standard chunking potentially offers

greater robustness when synthesizing information across multiple distinct document segments.



(a) Single-Document Questions (b) Multi-Document Questions

Figure 5.2: Comparison of Mean F1@20 across KB Types, Complexity, and Document Requirement (Threshold=0.4)

Further examining the effectiveness of the metadata embedding strategy (Hypothesis **H3**), Figure 5.3 highlights the difference in Precision@20 between the `Enhanced` and `Chunked` KBs. This confirms the nuanced outcome suggested by the F1 scores and average metrics (Table 5.3). The metadata provides a substantial precision improvement for complex single-document queries (+0.075) but leads to a slight degradation for complex multi-document queries (-0.05). This implies the metadata signal aids retrieval focus in specific complex scenarios but might occasionally introduce conflicting signals or ambiguity when broader synthesis across multiple documents is necessary, potentially interacting non-optimally with the content representation in the embedding space.

The recall dynamics (Figure 5.4) show that both structured KBs achieve significantly higher recall than the monolithic baseline, with recall generally increasing with retrieval depth K. The `Enhanced` KB consistently maintains a slight recall advantage over the `Chunked` KB, noticeable at K=20 and K=30, aligning with the average Recall@20 scores in Table 5.3. The relatively high MRR scores across all KBs suggest that when a relevant document is retrieved, it often appears early in the ranked list, despite overall recall limitations.

In synthesis, the ground-truth evaluation unequivocally validates the necessity of chunking for effective retrieval (H1 strongly affirmed). The comparison between standard and enhanced chunking reveals that the specific strategy of embedding concatenated metadata yields marginal average gains and notable precision benefits in specific complex single-document scenarios, but does not represent a universally superior approach (H3 partially supported, with significant caveats). Standard chunking demonstrates competitive performance and superior robust-

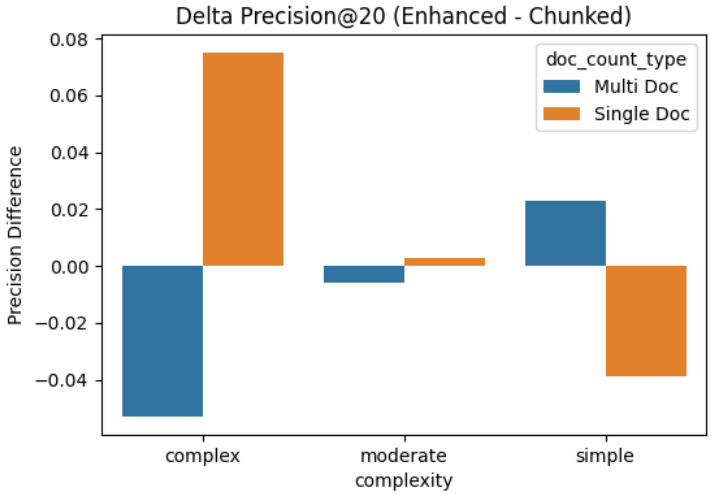


Figure 5.3: Difference in Precision@20 (Enhanced KB - Chunked KB) by Complexity and Document Requirement (Threshold=0.4)

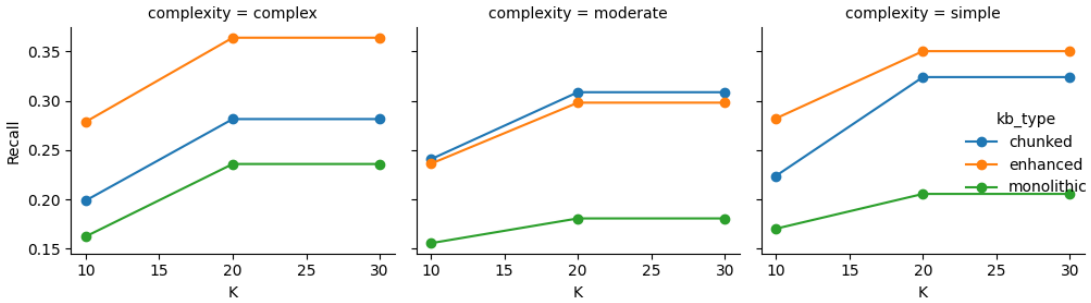
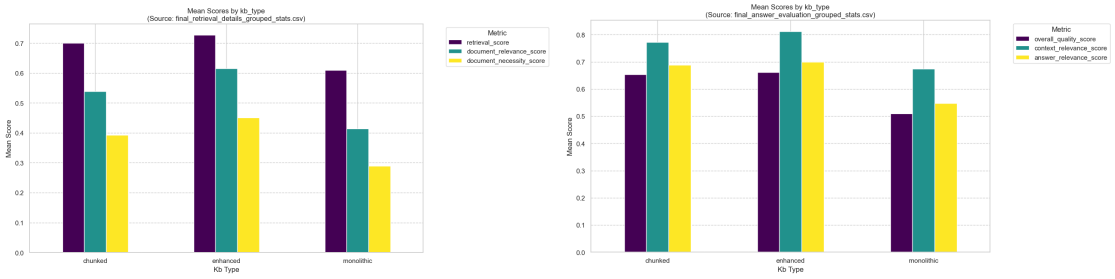


Figure 5.4: Average Recall vs. Retrieval Depth (K) by Complexity and KB Type (Threshold=0.4)

ness, particularly for multi-document F1 score, presenting a compelling default strategy. The choice between them involves a trade-off between optimizing for average recall (marginally favouring Enhanced) and ensuring robust performance across diverse query types (favouring Chunked).

5.6.2 LLM-Evaluated Context and Answer Quality

Complementing the ground-truth analysis, this subsection reports results from LLM-as-judge assessments across the entire 1,145-question dataset. These automated metrics serve as scalable proxies, offering insights into the perceived quality of the retrieved context, the necessity of individual retrieved documents, and the quality of the final generated answers. Figures 5.5a and 5.5b provide high-level visual comparisons of key mean scores discussed below.



(a) Mean Per-Document Retrieval Detail Scores

(b) Mean Context & Answer Evaluation Scores

Figure 5.5: Mean LLM-Evaluated Scores by KB Type

Context Quality Metrics (Per-Document Relevance & Necessity)

LLM evaluations first assessed the quality of the retrieved context documents individually before considering the overall answer. Key metrics include Document Relevance (is the document relevant to the query?) and Document Necessity (is the document needed to answer the query?). Table 5.4 presents the mean scores for these per-document metrics, averaged across all retrieved documents (up to 5 per query) for each KB type.

Consistent with ground-truth findings, the LLM judge perceives documents retrieved by the structured KBs as substantially more relevant and necessary than those from the Monolithic KB. The `Enhanced` KB achieves the highest mean scores for both Document Relevance (0.616) and Necessity (0.451), followed by the `Chunked` KB (Relevance: 0.539, Necessity: 0.394). The Monolithic KB lags sig-

Table 5.4: LLM-Evaluated Per-Doc Context Metrics (Mean, Scale: 0-1)

Metric	Monolithic	Chunked	Enhanced
Document Relevance Score	0.414	0.539	**0.616**
Document Necessity Score	0.290	0.394	**0.451**

nificantly (Relevance: 0.414, Necessity: 0.290), indicating its retrieved documents are often judged as less useful.

To understand the underlying score distributions, ridgeline plots visualize the estimated probability density for each KB type; peaks indicate more frequent scores, and the vertical dashed line marks the median. Figure 5.6 shows the distribution for Document Relevance scores. The structured KBs not only have higher mean scores but also distributions shifted towards higher relevance. The median for the `Enhanced` KB distribution is highest at 0.70, followed by the `Chunked` KB at 0.60, and finally the `Monolithic` KB at 0.45. This suggests the Enhanced KB more consistently retrieves documents perceived as highly relevant according to the LLM judge.

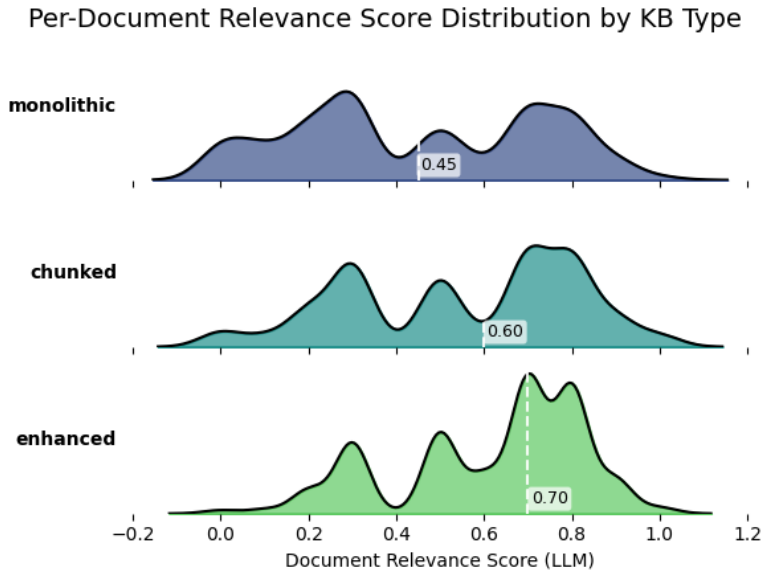


Figure 5.6: Per-Document Relevance Score Distribution by KB Type (LLM-Eval)

Similarly, the Document Necessity distributions (Figure 5.7) reveal differences beyond the mean. While all distributions are multi-modal (i.e., exhibiting multiple peaks), reflecting variability in how necessary individual documents are, the median necessity score progressively increases from Monolithic (0.25) to Chunked

(0.40) to Enhanced (0.50). This indicates that documents retrieved by the Enhanced KB are most frequently judged as necessary for constructing the answer, followed by Chunked, with Monolithic documents often deemed less essential.

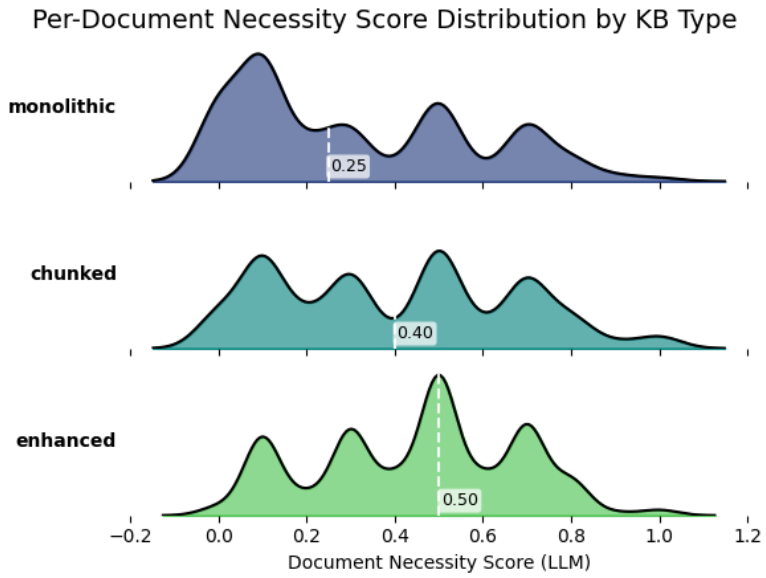


Figure 5.7: Per-Document Necessity Score Distribution by KB Type (LLM-Eval)

Overall, these per-document LLM evaluations reinforce the superiority of structured KBs in retrieving context deemed relevant and necessary, with the Enhanced KB showing an advantage in consistency according to the LLM judge.

Answer Quality Metrics

The quality of the final generated answer was assessed for Answer Relevance (to the original query), Faithfulness (to the provided context), and Completeness (based on the context). While these metrics directly evaluate the output of the generator LLM, they are analysed here as downstream indicators, reflecting the transitive impact of the quality and utility of the context retrieved from each Knowledge Base configuration. Average scores are shown in Table 5.5. As previously noted, citation quality scores were uniformly low and are omitted.

The LLM-evaluated answer quality metrics largely mirror the context quality trends and provide support for hypothesis **H2**. Answers generated using context from the `Chunked` (Overall: 0.653) and `Enhanced` (Overall: 0.662) KBs were rated significantly higher overall than those from the `Monolithic` KB (Overall: 0.509). This indicates that the superior context provided by structured KBs directly contributes to generated answers perceived as more relevant, faithful, and complete,

Table 5.5: LLM-Evaluated Answer Quality Metrics (Mean Scores, Scale: 0-1)

Metric	Monolithic	Chunked	Enhanced
Answer Relevance	0.547	0.688	**0.700**
Answer Faithfulness	0.667	0.745	**0.763**
Answer Completeness	0.453	**0.602**	**0.606**
Overall Quality Score	0.509	0.653	**0.662**

affirming H2.

The distribution of Answer Completeness scores (Figure 5.8) is particularly revealing. The Monolithic KB's distribution is broad with a lower median (0.60), while both Chunked and Enhanced share a higher median (0.70) and distributions concentrated towards higher completeness. This reinforces the interpretation that the *completeness* of the final answer is heavily constrained by the quality (relevance, necessity, sufficiency) of the retrieved context, which is demonstrably lower for the Monolithic KB.

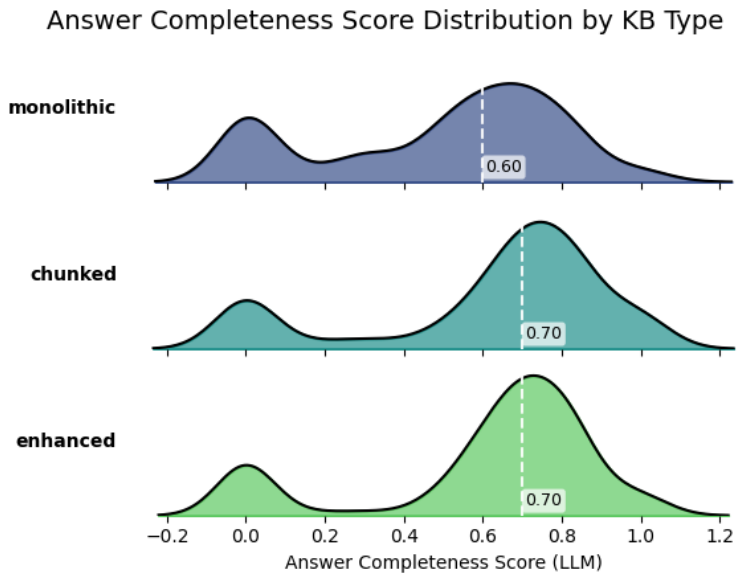


Figure 5.8: Answer Completeness Score Distribution by KB Type (LLM-Eval)

5.6.3 Operational Performance: Retrieval Latency

Beyond retrieval and generation quality, operational efficiency is a crucial consideration for practical RAG systems. This subsection addresses RQ2 by analyzing a key aspect of this efficiency: the time taken for the retrieval stage across the different KB configurations. Table 5.6 presents the mean retrieval latency.

Table 5.6: Retrieval Latency Comparison (Mean Values)

Metric	Monolithic	Chunked	Enhanced
Retrieval Latency (s)	0.764	**0.457**	0.473

The retrieval latency results reveal significant operational differences directly attributable to KB structure, pertinent to RQ2. The `Monolithic` KB exhibits markedly higher retrieval latency (mean 0.764s) compared to both structured KBs (`Chunked`: 0.457s, `Enhanced`: 0.473s). This disparity is likely due to the increased overhead associated with processing larger, unsegmented document representations during the initial candidate retrieval phase, even with subsequent truncation before final embedding.

Between the structured approaches, the standard `Chunked` KB demonstrates a slight edge in retrieval speed over the `Enhanced` KB. While the difference is small (approx. 16ms), it suggests that the process of handling and potentially concatenating metadata for the Enhanced KB introduces a minor latency overhead during retrieval compared to the simpler content-only approach of the standard Chunked KB.

Therefore, from a purely retrieval time perspective, structured KBs offer a substantial efficiency advantage over the monolithic approach. Standard chunking provides the fastest retrieval, although the enhanced approach is only marginally slower. This highlights a trade-off: the potential (though nuanced) quality improvements observed with the Enhanced KB in previous sections come at the cost of a small increase in retrieval latency compared to the standard Chunked KB.

5.7 Discussion and Synthesis

This section synthesizes the findings from ground-truth retrieval, LLM-evaluated quality, and operational metrics to provide a holistic comparison and directly address the research questions.

Overall, the empirical evidence strongly validates the importance of KB structure. Both `Chunked` and `Enhanced` KBs demonstrably surpassed the `Monolithic`

baseline in ground-truth retrieval effectiveness (H1 affirmed). The clear performance penalty incurred by the monolithic approach highlights the inadequacy of naive document handling, especially given embedding model limitations. LLM evaluations provide strong support for hypothesis **H2**, indicating that the superior context provided by structured KBs directly contributes to generated answers perceived as more relevant, faithful, and complete.

The comparison between `Chunked` and `Enhanced` KBs reveals a more nuanced picture regarding the effectiveness of embedding metadata directly (H3). While the `Enhanced` KB showed slightly higher average recall and NDCG in ground-truth tests and excelled in precision for complex single-document queries, the standard `Chunked` KB proved more robust in F1 score for multi-document queries. This suggests the metadata embedding strategy provided marginal, context-dependent benefits rather than a universally superior approach. Operationally, the trade-offs between retrieval/generation quality and computational efficiency are evident.

Based on this evidence:

- **RQ1 (KB Structure Impact):** Structure is critical. Chunked approaches (standard and enhanced) significantly outperformed monolithic retrieval, likely due to better focus and mitigation of truncation effects.
- **RQ2 (Performance vs. Overhead):** Trade-offs exist. The optimal choice regarding RQ2 depends heavily on whether minimizing latency or minimizing total token cost is the primary operational constraint.
- **RQ3 (Enhanced KB Effectiveness):** Embedding metadata directly with content yielded marginal and context-dependent improvements over standard chunking in our setup. The benefits observed (e.g., slight average recall increase, precision boost for complex single-doc queries) may not always outweigh the potential for noise or the robustness of standard chunking, particularly for multi-document tasks.

In conclusion, while standard semantic chunking provides a major performance uplift over monolithic KBs for technical documentation RAG, the specific strategy of enhancing chunks by embedding concatenated metadata offered only modest, situational advantages in this study. The robustness of standard chunking, especially for multi-document queries, makes it a strong default choice.

5.8 Evaluation Limitations

It is crucial to acknowledge limitations inherent in the evaluation process, which could influence the interpretation of findings:

- **Ground Truth Generation Challenges:**
  - **LLM Reliability in Annotation Support:** Reliance on LLMs for initial relevance score suggestions introduces potential biases. While human validation provides the final judgment, initial suggestions might subtly influence annotators or miss edge cases.
  - **Human Annotator Consistency:** Subjectivity in relevance judgments, even with guidelines, can lead to inter-annotator disagreement. While only a single annotator was used for this project due to resource constraints, this introduces a potential single point of bias.
  - **Relevance Scale Granularity:** Using a continuous scale aims for nuance but can be harder to apply consistently than discrete labels (e.g., relevant, not relevant). The chosen threshold (0.4) for binary metrics directly impacts results.
- **LLM-as-Judge Subjectivity:** The metrics evaluating context and answer quality rely on LLM judgments (e.g., context relevance, answer faithfulness). These judgments, while scalable, inherit the biases and limitations of the evaluator LLM and may not perfectly align with human perception of quality [Zheng et al., 2023, Owens et al., 2024, Verga et al., 2024].
- **Evaluation Dataset Scope:** The evaluation dataset, while diverse, represents a sample of possible queries. Performance might differ on unseen query types or topics not well-represented in the source documentation or evaluation set.
- **Completeness Assessment:** Judging the **completeness** of an answer is inherently difficult, especially for complex queries. The LLM judge assesses completeness relative to the *provided context*, not absolute completeness, which is an important distinction.

These limitations underscore the importance of interpreting results holistically, considering the complementary nature of the different metric types employed.

Chapter 6

Conclusion and Future Work

6.1 Synthesis of Findings and Critical Reflection

This project empirically investigated the critical role of Knowledge Base (KB) structure in Retrieval-Augmented Generation (RAG) systems applied to complex, evolving technical documentation, using HarmonyOS ArkTS guides as a case study (Chapter 1). The primary motivation was to address known LLM deficiencies by grounding generation in external knowledge, requiring an understanding of optimal KB design (Chapter 2). We compared Monolithic, standard Chunked, and metadata-Enhanced chunking strategies within a consistent two-stage RAG pipeline (Section 5.2).

The findings unequivocally underscore the inadequacy of naive Monolithic KB structures for this task. Structured approaches (Chunked and Enhanced) demonstrated substantial performance gains; for instance, the Chunked KB yielded approximately 13.4% higher NDCG@10 scores (0.585 vs 0.516) compared to the Monolithic baseline in ground-truth evaluation (Section 5.6.1, Table 5.3). This performance delta likely stems from mitigating severe document truncation necessitated by embedding model limits (affecting 67% of monolithic documents, Section 4) and enabling more focused semantic retrieval compared to the diluted representations of large, multi-topic documents (Section 5.6.1). The improved context quality from structured KBs directly correlated with higher LLM-evaluated answer relevance, faithfulness, and completeness, confirming that retrieval quality is a crucial bottleneck for downstream generation (Section 5.6.2).

The comparison between standard Chunking and embedding concatenated metadata (Enhanced KB) revealed a context-dependent trade-off rather than universal superiority for the latter (Section 5.6.1, Section 5.6.2). While metadata concatenation provided a precision uplift for complex *single-document* queries (approx. +0.075 P@20) and marginal average recall gains (Table 5.3), it led to per-

formance degradation for complex *multi-document* queries (approx. -0.05 F1@20) compared to standard chunking. **Critically**, this suggests the chosen enhancement strategy, simple concatenation pre-embedding, may introduce complexities. The BGE embedding model, despite its general capabilities, might struggle to effectively disentangle and leverage the concatenated metadata signals, potentially treating them as distributional noise or failing to resolve conflicting metadata points during the synthesis required for multi-document queries (Section 5.6.1). This inherent limitation of the *chosen method* restricts its utility; alternative integration strategies might prove more robust (Section 5.8). Furthermore, standard Chunking offered marginally faster retrieval latency (0.457s vs 0.473s), presenting a compelling baseline due to its effectiveness and efficiency (Table 5.6).

Therefore, the core takeaway is not merely that structure matters, but that *how* structure and metadata are implemented involves critical trade-offs impacting retrieval precision, multi-document synthesis, and operational efficiency. The simplistic concatenation approach, while tested, showed limited and context-specific benefits in this implementation.

6.2 Contributions

Based on the empirical evidence derived specifically from the ArkTS domain, this research offers the following contributions:

1. **Quantified Impact of Chunking:** Provided specific quantitative evidence demonstrating the necessity and benefit of semantic chunking over monolithic approaches for technical RAG, showing significant improvements (e.g., NDCG@10, Recall@20) within a defined RAG architecture (Section 5.6.1, Table 5.3).

2. **Evidence on Concatenated Metadata Embedding:** Offered empirical results on the *specific strategy* of concatenating metadata pre-embedding, revealing its nuanced and limited effectiveness (precision gains in specific scenarios offset by F1 score degradation in others). This contributes data to the ongoing discussion about optimal metadata integration in RAG, highlighting potential pitfalls of this particular method, possibly due to embedding model limitations in disentangling mixed signals.

3. **Domain-Specific Evaluation Methodology:** Implemented and documented a multi-faceted evaluation framework tailored for technical documentation RAG where ground truth is scarce. A key component was a *semi-automated annotation workflow* leveraging LLM suggestions with human verification, balancing scalability needs with judgement accuracy (Section 5.4). While constituent parts (LLM-as-judge, GT subsets) are known,

the specific workflow represents a practical contribution for similar resource-constrained projects.

4. **Evidence-Based Guidance for ArkTS RAG:** Provided actionable insights grounded in performance data for the HarmonyOS ArkTS domain, establishing standard semantic chunking as a highly effective and robust baseline (Section 5.7), while cautioning that the tested metadata enhancement yielded only situational benefits potentially outweighed by its complexity and multi-doc synthesis challenges (Section 5.7).

6.3 Strategic Directions for Future Work

The findings and identified limitations (Section 5.8) motivate several strategic directions for advancing RAG systems in technical domains:

1. **Prioritising Metadata Integration Research:** Given the mixed results of metadata concatenation (Section 5.7), systematically comparing this with alternative approaches (e.g., dedicated metadata filtering layers, separate graph-based metadata representations, multi-vector approaches, referenced in Chapter 2) is crucial to determine truly effective strategies for leveraging document context beyond raw text. This addresses the limitation of our chosen concatenation method (Section 5.8).

2. **Developing Adaptive Retrieval Strategies:** Standard chunking proved robust, but exploring adaptive methods remains unexplored here. This includes investigating adaptive chunk sizing based on semantic density (Chapter 2) and, critically, hybrid retrieval techniques combining dense search with keyword-aware sparse methods (e.g., BM25, Chapter 2). The latter directly addresses a known weakness of pure dense retrieval in handling specific technical terms or code identifiers, frequently essential in developer documentation.

3. **Investigating Model Interactions:** The interplay between KB structure and the specific embedding/generation models requires deeper study (Chapter 2). Future work should benchmark performance with different embedding models (e.g., those explicitly trained for code or instruction-following) to see if they better handle structured input like concatenated metadata, and assess how different generator LLMs consume and synthesise retrieved context from various KB structures.

4. **Mechanistic Interpretability of Embedding Models:** Use mechanistic interpretability to analyse how the BGE embedding model internally processed concatenated metadata, aiming to explain the Enhanced KB's nu-

anced performance (Section 5.6.1). This could reveal if metadata acted as signal or noise, guiding better integration strategies, though it may require significant computational resources more suited to industry research.

Strategically, the most pressing needs arising from this work involve developing more sophisticated metadata integration techniques beyond simple concatenation and exploring adaptive/hybrid retrieval to combine semantic understanding with keyword precision. Addressing these, alongside more robust evaluation, represents a clear roadmap for building more powerful and reliable RAG systems for complex technical knowledge work.

Bibliography

Acceldata Blog. Metadata example: Types, applications, and importance in data management. Acceldata Blog, N/A. URL https://www.acceldata.io/blog/metadata-example-types-applications-and-importance-in-data-management. Accessed: 2025-04-10. Year estimated circa 2023 based on content.

Omar Ahmed, Usman Abbasi, Faraz Ahmed, Rabia Naeem, and Muhammad Usman Bashir. Large language models and hallucinations: A new challenge for medical practice. *Cureus*, 16(12):e50637, 2024. doi: 10.7759/cureus.50637. URL https://www.cureus.com/articles/204178-large-language-models-and-hallucinations-a-new-challenge-for-medical-practice.

Yujia Bao, Hongguang Li, Xiaoya Li, and Jingdong Zhou. Chunking-free in-context retrieval for long-context question answering. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 814–829, Bangkok, Thailand, 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.71. URL https://aclanthology.org/2024.acl-long.71.pdf.

Yue Cao, Yiming Shen, Zhiqing Sun, Tanmoy Chakraborty, and Kai-Wei Chang. Retrieval-augmented generation for large language models: A survey. Technical Report 2405.06211v1, arXiv, 2024. URL https://arxiv.org/abs/2405.06211v1.

Chroma Research. Evaluating chunking strategies for retrieval. Chroma Research Blog, N/A. URL https://research.trychroma.com/evaluating-chunking. Accessed: 2025-04-10. Year estimated circa 2024 based on content.

Daniel E. Dahl, Terry Matthews, Terry Yue Chen, Neel Guha, and John J. Nay. Large legal fictions: Profiling hallucinations in large language models. Technical Report 2401.01301, arXiv, 2024. URL https://arxiv.org/abs/2401.01301. Accessed URL from original query: https://nacmnet.org/wp-content/uploads/Dahl-et-al-Large-Legal-

Fictions-Hallucinations-in-LLMs-2401.013012024JAN02-37pp.pdf. Verified via arXiv.

Shahul Es, Jithin James, Anze Saulnier, Ranjit Rajagopal, and Philip Avvaru. RAGAS: Automated evaluation of retrieval augmented generation. Technical Report 2309.15217v1, arXiv, 2023. URL https://arxiv.org/abs/2309.15217v1. Original URL pointed to a non-archival PDF copy.

Xin Jiang, Tianrui Li, Zhaohui Jiang, Ningyu Zhang, and Huajun Chen. Optimal RAG practices: A comprehensive study of retrieval-augmented generation. Technical Report 2407.01219v1, arXiv, 2024. URL https://arxiv.org/abs/2407.01219v1.

Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 7(3):535–547, 2019. Also available as arXiv preprint arXiv:1702.08734.

Tian Lan, Hangyu Li, Wenhao Huang, Zhenhuan Zhang, Junyu He, Xingxuan Li, Jiajun Bu, and Jianfeng Feng. A survey on retrieval-augmented generation for large language models. Technical Report 2407.13193v1, arXiv, 2024. URL https://arxiv.org/abs/2407.13193v1.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems 33 (NeurIPS 2020)*, pages 9459–9474. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/6b493230205f780e1bc26945df7481e5-Paper.pdf.

Huayang Li, Pichao Wang, Fan Wang, Tianlong Chen, and Zhangyang Wang. Retrieval-augmented generation for computer vision: A survey. Technical Report 2503.18016v1, arXiv, 2025. URL https://arxiv.org/abs/2503.18016v1.

Tianle Li, Chongyang Bai, Tianyi Zhang, Zhengyang Liu, Ningyu Zhang, and Huajun Chen. Know your RAG: Dataset taxonomy and generation strategies for evaluating RAG systems. Technical Report 2411.19710v1, arXiv, 2024a. URL https://arxiv.org/abs/2411.19710v1.

Tianle Li, Chongyang Bai, Tianyi Zhang, Zhengyang Liu, Ningyu Zhang, and Huajun Chen. RAG playground: An evaluation framework for retrieval-augmented generation strategies. Technical Report 2412.12322, arXiv, 2024b.

URL https://arxiv.org/abs/2412.12322. Also listed as ref46 with version v1.

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. Technical Report 2307.03172v1, arXiv, 2023. URL https://arxiv.org/abs/2307.03172v1.

Andrea Matarazzo and Riccardo Torlone. A survey on large language models with some insights on their capabilities and limitations. Technical Report 2501.04040, arXiv, 2025. URL https://arxiv.org/abs/2501.04040. Version 1.

Maxim AI Blog. Evaluating RAG performance: Metrics and benchmarks. Maxim AI Blog, N/A. URL https://www.getmaxim.ai/blog/rag-evaluation-metrics/. Accessed: 2025-04-10. Year estimated circa 2024 based on content.

Microsoft Learn. RAG solution design and evaluation guide. Microsoft Learn Documentation, nodate. URL https://learn.microsoft.com/en-us/azure/architecture/ai-ml/guide/rag/rag-solution-design-and-evaluation-guide. Accessed: 2025-04-10.

Andrew D. Owens, Anson Ho, Tejas Gokhale, and Pradeep Ravikumar. A multi-LLM debiasing framework. Technical Report 2409.13884v1, arXiv, 2024. URL https://arxiv.org/abs/2409.13884v1.

Prem Blog. Chunking strategies in retrieval-augmented generation (RAG) systems. Prem Blog, N/A. URL https://blog.premai.io/chunking-strategies-in-retrieval-augmented-generation-rag-systems/. Accessed: 2025-04-10. Year estimated circa 2023-2024 based on content.

Prompting Guide. Retrieval augmented generation (RAG). Prompting Guide Website, nodate. URL https://www.promptingguide.ai/techniques/rag. Accessed: 2025-04-10.

Nils Reimers and Iryna Gurevych. Sentence-BERT: Sentence embeddings using siamese BERT-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3982–3992, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1410. URL https://aclanthology.org/D19-1410/.

Zhiqing Sun, Yue Cao, Yiming Shen, Jingbo Zhou, and Jiawei Han. Agentic RAG: A survey on retrieval-augmented generation with autonomous agents.

Technical Report 2501.09136v1, arXiv, 2025. URL https://arxiv.org/abs/2501.09136v1.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017. URL http://arxiv.org/abs/1706.03762.

Philipp Verga, Pontus Stenetorp, Hwaran Lee, Jo Kristian Bergum, and Jonas Gehring. Replacing judges with juries: Evaluating LLM generations with a panel of diverse models. Technical Report 2404.18796v1, arXiv, 2024. URL https://arxiv.org/abs/2404.18796v1.

Gaurav Verma, Imam Munir, Muhammad Umair, Muhammad Asif, Muhammad Usman, Muhammad Muzammal, Muhammad Amjad, Muhammad Waqas, Muhammad Attique Khan, and Muhammad Sharif. Retrieval-augmented generation for large language models: A survey. Technical Report 2312.10997v5, arXiv, 2023. URL https://arxiv.org/abs/2312.10997v5.

Ellen M. Voorhees and Donna K. Harman. Overview of TREC 2001. In *Proceedings of the Tenth Text REtrieval Conference (TREC 2001)*, Gaithersburg, MD, USA, 2001. National Institute of Standards and Technology (NIST). URL https://trec.nist.gov/pubs/trec10/papers/overview_10.pdf.

Boxin Wang, Weijia Shi, Julian Martin Eisenschlos, Pengfei Liu, and Noah A. Smith. Knowledge overshadowing: Investigating the impact of knowledge conflicts on factual hallucination in large language models. Technical Report 2502.16143v1, arXiv, 2025. URL https://arxiv.org/abs/2502.16143v1.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems 35 (NeurIPS 2022)*, pages 24824–24837. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/9d5609613524ecf4f15af0f7b31abca4-Paper-Conference.pdf.

Yifei Yuan, Qingqing Chen, Zijun Yao, Jiaming Shen, and Hai Zhao. GraphRAG: A survey on retrieval-augmented generation with knowledge graphs. Technical Report 2501.13958, arXiv, 2025. URL https://arxiv.org/abs/2501.13958.

Yiming Zhang, Yiyun Zhao, Yifan Gao, Jingfeng Yang, and Meng Jiang. Meta knowledge for retrieval augmented large language models. Technical Report 2408.09017v1, arXiv, 2024a. URL https://arxiv.org/abs/2408.09017v1.

Zhenhuan Zhang, Jiajun Bu, and Jianfeng Feng. Retrieval-augmented generation systems: Automatic dataset creation, evaluation and boolean agent setup. Technical Report 2403.00820v1, arXiv, 2024b. URL https://arxiv.org/abs/2403.00820v1.

Zhenhuan Zhang, Jiajun Bu, and Jianfeng Feng. Optimizing and evaluating enterprise retrieval-augmented generation (RAG): A content design perspective. Technical Report 2410.12812v1, arXiv, 2024c. URL https://arxiv.org/abs/2410.12812v1.

Zhenhuan Zhang, Jiajun Bu, and Jianfeng Feng. S2 chunking: A hybrid framework for document segmentation through integrated spatial and semantic analysis. Technical Report 2501.05485v1, arXiv, 2025a. URL https://arxiv.org/abs/2501.05485v1.

Zhenhuan Zhang, Jiajun Bu, and Jianfeng Feng. Rethinking document structure representation for legal information retrieval. Technical Report 2502.10868v1, arXiv, 2025b. URL https://arxiv.org/abs/2502.10868v1.

Zhenhuan Zhang, Tian Lan, Jiajun Bu, and Jianfeng Feng. Emulating retrieval augmented generation via prompt engineering for enhanced long context comprehension in LLMs. Technical Report 2502.12462v1, arXiv, 2025c. URL https://arxiv.org/abs/2502.12462v1.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging LLM-as-a-judge with MT-Bench and chatbot arena. Technical Report 2306.05685v2, arXiv, 2023. URL https://arxiv.org/abs/2306.05685v2.

Xin Zhou, Jiajie Zhang, Hailiang Huang, Jianfeng Qu, Weidong Bao, and You Zhou. CHUNKRRAG: A novel LLM-chunk filtering. OpenReview submission, N/A. URL https://openreview.net/forum?id=NsvaW3Y6Su. Accessed: 2025-04-10. Status and year unclear from source.