Citation-Aware Retrieval-Augmented Generation (RAG) Application

A Production-Oriented System for Verifiable Question Answering over PDF Documents

Author:
Anusara Senanayaka

Role / Focus:
AI Systems · Retrieval Engineering · LLM Applications

Technologies:
Python · Llama 3.2 (3B) · ChromaDB · HNSW · BM25 · MiniLM · RAGAS

Project Type:
Professional Portfolio Project

Date:
December 2025

Table of Contents

Table of figures

Executive Summary

This project presents a Citation-Aware Retrieval-Augmented Generation (RAG) Application specialized in answering real-world PDF documents, e.g., legal policies, research papers and internal technical documentation. The system emphasizes making accurate, reliable answers with explicit sources references in contrast to the flaws incurred by standard RAG demos.

Most prior RAG works build on naive document chunking, weak understanding from PDFs and/or purely semantic retrieval and suffer from loss of document structure during encoding; inaccurate or irrelevant context retrieval and ungrounded/hallucinated answers. This system was developed to mitigate these problems by focusing on structure aware document ingestion, semantic chunking, hybrid retrieval strategies and citation coherence.

The architecture is modular and production-oriented, designed from the ground-up to run in resource constrained environments with small models and efficient approximated search. Thus, the system trades performance for accuracy and practicability and is deployable in real applications rather the laboratory experiments.

Problem Statement & Motivation

Retrieval-Augmented Generation has been recently proposed to address this problem for large language models: it aims to answer questions given external context from open-domain documents. However, most RAG systems have difficulties to cope with complex PDF documents. Such shortcomings are most evident in legal, academic or enterprise documentation where structure, precision and traceability is required.

- Common challenges observed in existing RAG pipelines include:
- Loss of logical reading orders when extracting text from PDFs
- Poor handling of tables and structured content
- Fixed-size chunking that breaks semantic coherence
- Retrieval methods that fail to match exact identifiers, clauses, or numeric references
- Lack of reliable citation mechanisms to verify generated answers

These limitations significantly reduce trust in generated responses, particularly in high-stakes environments where users must verify information against original sources. The motivation behind this project was to design an RAG system that treats **document understanding and citation accuracy as first-class concerns**, rather than secondary features.

System Objectives

The primary objectives of the Citation-Aware RAG Application are:

- Accurately ingest and normalize complex PDF documents while preserving structure
- Extract both textual and tabular information in a usable format
- Perform semantic-aware document chunking based on content similarity
- Support hybrid retrieval using both semantic vector search and exact keyword matching
- Generate answers that are explicitly grounded in retrieved document content
- Attach clear source and page-level citations to every response
- Operate efficiently on consumer-grade hardware without relying on large-scale infrastructure

These objectives guided all architectural and technological decisions throughout system design.

Key System Capabilities

The system provides the following core capabilities:

- **Structure-aware PDF ingestion** that preserves reading order and removes noise such as headers and footers
- **Table extraction and normalization**, converting tabular data into Markdown for improved retrieval
- **Semantic chunking** using sentence-level embeddings to identify natural topic boundaries
- **Hybrid retrieval** combining approximate vector search (HNSW) with BM25 keyword search
- **Optional Query enhancement** using HyDE and cross-encoder re-ranking to improve retrieval relevance
- **Citation-aware answer generation**, enabling traceable and verifiable outputs
- **Modular architecture**, allowing individual components to be extended or replaced independently

Together, these capabilities enable reliable question answering over complex documents where accuracy and trust are essential.

High-Level System Architecture

Architecture Overview

The Citation-Aware RAG Application follows a modular, pipeline-based architecture where each component is responsible for a single, well-defined concern. This design improves clarity, maintainability, and extensibility, while also allowing individual components to be optimized independently.

At a high level, the system is composed of three major stages: document ingestion, storage and indexing, retrieval and generation. Data flows sequentially through these stages, ensuring that document structure and metadata are preserved throughout the pipeline.

System Architecture diagram

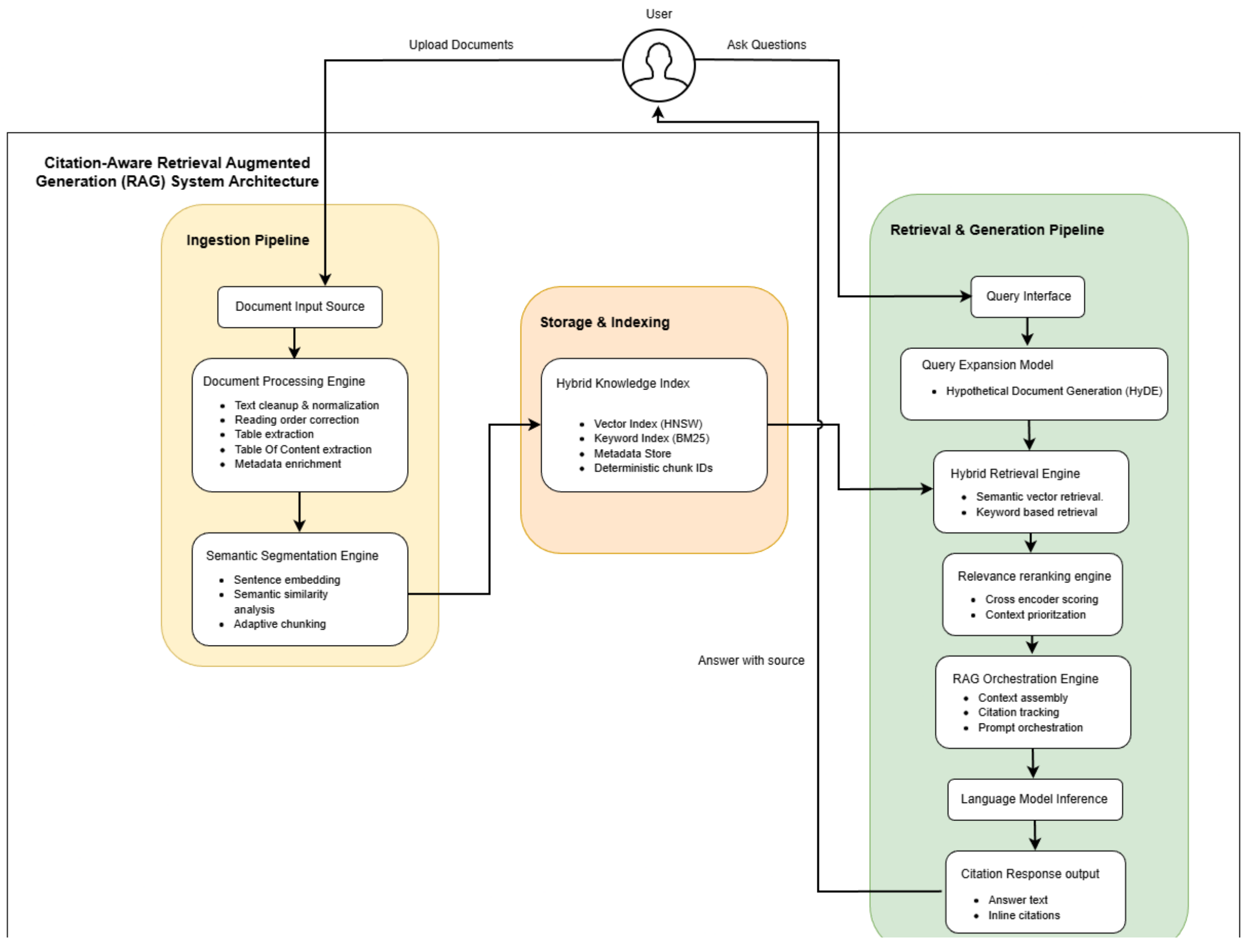

Figure 1 System Architecture diagram

End-to-End Data Flow

1. Source PDF documents are ingested and normalized by the document ingestion layer.
2. Cleaned documents are semantically segmented into coherent chunks.
3. Chunks are embedded and stored in a vector database along with enriched metadata.
4. User queries are enhanced and processed by the query engine.
5. Relevant chunks are retrieved using hybrid retrieval strategies.
6. Retrieved context is assembled and passed to the language model.
7. The final response is generated with explicit source and page-level citations.

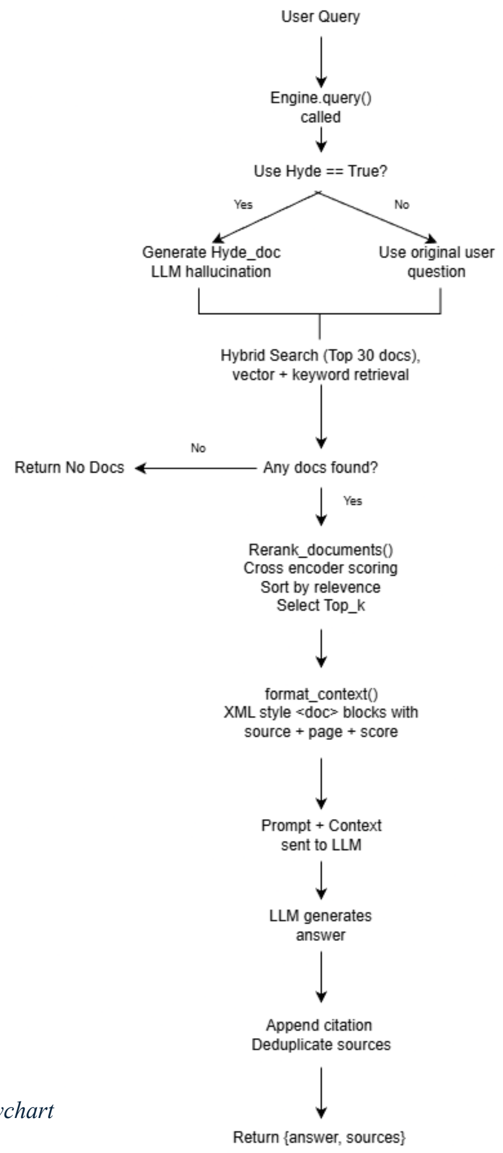
User Query
Engine.query() called
Use Hyde == True?
Yes
No
Generate Hyde_doc LLM hallucination
Use original user question
Hybrid Search (Top 30 docs), vector + keyword retrieval
No
Return No Docs
Any docs found?
Yes
Rerank_documents() Cross encoder scoring Sort by relevence Select Top_k
format_context() XML style <doc> blocks with source + page + score
Prompt + Context sent to LLM
LLM generates answer
Append citation Deduplicate sources
Return {answer, sources}

Figure 2 End to End data flowchart

System Design & Component Breakdown

Configuration & Core Types

The system uses a centralized configuration module (Config.py) to manage all tunable parameters, such as model selections, chunking thresholds, and retrieval settings. This approach avoids hard-coded values and enables rapid experimentation and configuration changes without modifying core logic.

Core data representations are defined in Types.py. These structured types represent documents and chunks, ensuring consistent handling of content and metadata across all stages of the pipeline. Each chunk encapsulates both textual content and associated metadata required for retrieval and citation.

Document Ingestion Layer

The document ingestion layer (loader.py) is responsible for transforming raw PDF files into clean, structured, and machine-readable documents. This functionality is implemented in the PDF loader component.

Key processing steps include:

- **Header and footer removal** by ignoring the top and bottom 50 pixels of each page
- **Noise reduction** to discard irrelevant visual and textual artifacts
- **Reading order normalization**, ensuring content is processed top-to-bottom and left-to-right
- **Table extraction**, where tabular content is converted into Markdown format
- **Table of Contents (TOC) handling**, allowing high-level document structure to be captured
- **Metadata enrichment**, including deterministic document IDs, source references, and page numbers

During ingestion, each page is processed sequentially. Text blocks and tables are extracted, sorted for logical flow, merged across pages where necessary, and empty pages are skipped. The final output is a structured document object enriched with metadata and ready for semantic processing.

Semantic Chunking Strategy

Rather than relying on fixed-size text splitting, the system employs a semantic chunking strategy to preserve thematic coherence. This approach is implemented in the document splitter component (splitter.py).

The process begins by generating sentence-level embeddings using a lightweight and efficient embedding model (MiniLM-L6). Cosine similarity is then calculated between consecutive sentences to identify shifts in semantic meaning. Chunk boundaries are determined using a percentile-based thresholding strategy, allowing the system to adaptively segment documents based on content rather than arbitrary length constraints.

This method results in chunks that align more closely with human-understandable sections, improving both retrieval accuracy and answer grounding.

Ingestion Pipeline Orchestration

The ingestion pipeline component (pipeline.py) orchestrates the end-to-end flow between document loading, semantic chunking, and vector storage. It ensures that documents are consistently processed using the same configuration and that all generated chunks are embedded and stored correctly.

By separating orchestration logic from individual processing steps, the pipeline maintains a clean separation of concerns and simplifies future extensions or modifications to the ingestion workflow.

Vector Storage & Retrieval Layer

The retrieval layer is implemented using a hybrid search strategy to balance semantic understanding with lexical precision (vector_store.py).

- **HNSW vector search** is used as the primary semantic retrieval mechanism, providing fast approximate nearest-neighbor search with high accuracy.
- **BM25 keyword search** complements semantic retrieval by enabling exact matching for identifiers, legal clauses, and numeric references.

This hybrid approach improves recall and precision across a wide range of query types, particularly in domains where exact wording is critical.

Query Enhancement & Re-Ranking

To further improve retrieval quality, the system applies query enhancement and re-ranking techniques (modifiers.py). HyDE (Hypothetical Document Embeddings) is used to generate a plausible answer representation based on the user query. This representation is embedded and used as a proxy query, improving alignment between the query and relevant document content.

After initial retrieval, a cross-encoder is used to compare queries directly against retrieved chunks, allowing for more precise relevance scoring and re-ordering. These techniques are applied selectively to enhance accuracy while avoiding unnecessary computational overhead.

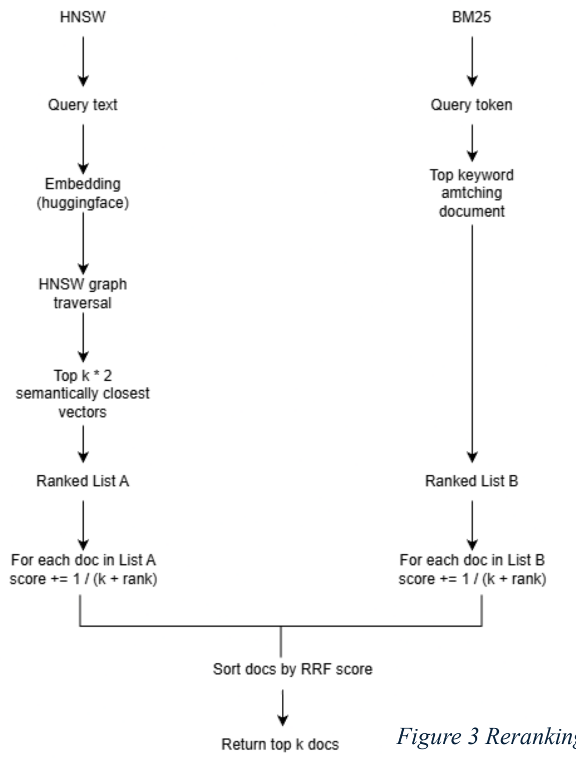
HNSW
BM25
Query text
Query token
Embedding (huggingface)
Top keyword amtching document
HNSW graph traversal
Top k * 2 semantically closest vectors
Ranked List A
Ranked List B
For each doc in List A score += 1 / (k + rank)
For each doc in List B score += 1 / (k + rank)
Sort docs by RRF score
Return top k docs

Figure 3 Reranking flowchart

Query Execution Engine

The query execution engine acts as the runtime coordinator of the system. It receives user queries, applies query enhancement strategies, retrieves relevant document chunks using hybrid retrieval, and assembles contextual information for generation.

Crucially, the engine preserves metadata throughout the entire process, ensuring that all generated responses include explicit citations referencing the original source documents and page numbers. This guarantees that answers are not only fluent but also verifiable and grounded in retrieved evidence.

Citation & Metadata Design

A core goal of this system is to ensure that every generated answer is **traceable, verifiable, and grounded in source documents**. To achieve this, citation handling is treated as a first-class concern throughout the retrieval and generation pipeline rather than as a post-processing step.

Each document chunk generated during ingestion carries **enriched metadata**, including a unique identifier, source document reference, and page number. This metadata is preserved across semantic chunking, vector storage, retrieval, and query execution stages.

During query execution, retrieved chunks are assembled into the context passed to the language model along with their associated metadata. The query engine ensures that citations are attached at the chunk level, enabling the final response to explicitly reference the original document source and page numbers.

This design provides several advantages:

- Generated answers can be verified directly against original documents
- Users can trace specific claims back to exact pages
- Hallucination risks are reduced by grounding responses in retrieved evidence
- The system becomes suitable for domains requiring high trust, such as legal or technical documentation

By embedding citation awareness directly into the system architecture, the application prioritizes transparency and reliability over purely fluent output.

Design Decisions & Trade-offs

The system was designed under practical constraints, with each major technical choice guided by performance, reliability, and real-world usability considerations. The following key decisions highlight the trade-offs made during development.

Lightweight Embedding Model Selection

Decision:
A lightweight sentence embedding model (MiniLM-L6) was used for semantic chunking and retrieval.

Reason:
The system is intended to operate efficiently on consumer-grade hardware, where larger embedding models may introduce excessive latency and memory usage.

Trade-off:
While larger models may provide richer semantic representations, this limitation is mitigated through hybrid retrieval and re-ranking strategies.

Semantic Chunking Over Fixed-Size Splitting

Decision:
Documents are segmented using semantic similarity rather than fixed token or character lengths.

Reason:
Fixed-size chunking often breaks semantic continuity and degrades retrieval quality, especially in long or structured documents. This method creates clusters of text that align closely with the thematic sections of the original document

Trade-off:
Semantic chunking introduces additional computation during ingestion, but this cost is incurred only once and significantly improves downstream retrieval accuracy.

Hybrid Retrieval Strategy

Decision:
The system combines HNSW-based vector search with BM25 keyword search.

Reason:
Pure semantic retrieval performs poorly for exact identifiers, legal clauses, and numeric references, while keyword search alone lacks contextual understanding.

Trade-off:
Maintaining two retrieval strategies increases system complexity, but the resulting improvement in precision and recall justifies the design.

Metadata-Centric Architecture

Decision:
Metadata is preserved and propagated across all system layers.

Reason:
Citation accuracy and traceability depend on consistent metadata handling.

Trade-off:
Additional metadata management slightly increases storage overhead but is essential for citation integrity.

Large Language Model (LLM) Selection

Decision:
The system uses **Llama 3.2 (3B parameters)** as the primary language model for answer generation.

Reason:
The project was designed to operate under practical hardware constraints while still delivering coherent, context-aware responses. Llama 3.2 (3B) provides a strong balance between language understanding capability and computational efficiency, making it suitable for local or resource-limited environments. Its open-weight nature also allows full control over deployment without reliance on external APIs.

Trade-off:
Compared to larger proprietary or open-source models, a 3B-parameter model has reduced reasoning depth and may struggle with highly complex multi-hop queries. This limitation is mitigated by strengthening the retrieval pipeline—through semantic chunking, hybrid search, and re-ranking—so that the model operates on high-quality, well-grounded context rather than relying solely on parametric knowledge.

Evaluation and Results

To assess the effectiveness of the Citation-Aware RAG Application, a **custom deterministic evaluation pipeline** was implemented and executed on a controlled dataset. The evaluation focuses on retrieval quality, citation correctness, answer relevance, and runtime performance, ensuring that results are reproducible and auditable.

All evaluation outputs are exported to a structured CSV report to support traceability and inspection.

Evaluation Setup

The evaluation pipeline measures system performance across the following dimensions:

- **Retrieval effectiveness**, ensuring relevant context is successfully retrieved
- **Ranking quality**, verifying that correct chunks appear at the top of results
- **Citation compliance**, confirming that answers reference valid sources
- **Answer relevance**, measuring semantic alignment between answers and expected content
- **Latency**, capturing end-to-end response time using a local 3B model

Evaluation was performed deterministically to eliminate randomness and allow consistent comparison across runs.

Aggregate Evaluation Results

Metric	Result
Retrieval Hit Rate	100%
Mean Reciprocal Rank	1.000
Citation Compliance	100%
Negative constraint Adherence	100%
Average Latency	15.05 s
Answer Relevancy	0.934

These results indicate that the system consistently retrieves correct context, ranks it at the top, and produces answers that remain grounded in cited sources.

Breakdown by Question Type

To further validate robustness, the evaluation dataset included multiple question categories.

Question Type	Hit rate	MRR	Citation Compliance	Relevancy
Reasoning	100%	1.000	100%	0.964
Simple	100%	1.000	100%	0.918
Technical	100%	1.000	100%	0.907

The system performs consistently across reasoning-heavy, simple factual, and technical queries, with strong relevance scores even for complex reasoning tasks.

Conclusion

This project demonstrates a production-oriented approach to building a **Citation-Aware Retrieval-Augmented Generation system** capable of answering questions over complex PDF documents with verifiable, source-grounded outputs.

By focusing on structure-aware document ingestion, semantic chunking, hybrid retrieval, and explicit citation handling, the system addresses key limitations found in many existing RAG implementations. Rather than relying on large models or brute-force scaling, the design emphasizes modularity, efficiency, and engineering judgment.

The resulting architecture is transparent, extensible, and suitable for real-world applications where accuracy and trust are essential. This project reflects an engineering mindset that prioritizes correctness, explainability, and practical deployability, making it a strong foundation for future enhancements and domain-specific adaptations.