Advanced RAG System

Technical Documentation & Testing Guide

Version 2.0 - February 2026

Author:	AI Research Team
Department:	Machine Learning Division
Document Type:	Technical Specification
Classification:	Internal Testing

Table of Contents

1. Introduction to Modular RAG Systems

Retrieval-Augmented Generation (RAG) represents a paradigm shift in how we approach natural language processing and information retrieval. By combining the power of large language models with efficient document retrieval mechanisms, RAG systems enable more accurate, contextual, and reliable responses to user queries.

The modular architecture of this system allows for flexible configuration and easy extension. Each component—from document loading and chunking to embedding generation and vector storage—can be independently configured, tested, and optimized. This modularity is crucial for maintaining system quality and adapting to evolving requirements.

Key Features:

- Multiple embedding provider support (OpenAI, Azure, Ollama, Sentence Transformers)
- Flexible chunking strategies with metadata preservation
- Hybrid search combining dense and sparse retrieval
- Advanced reranking with cross-encoder models
- Comprehensive observability and evaluation framework
- Production-ready error handling and logging

2. System Architecture Overview

The system follows a layered architecture with clear separation of concerns. At the foundation, we have the data ingestion layer responsible for loading various document formats (PDF, DOCX, TXT, HTML) and converting them into a standardized internal representation. This layer handles document parsing, text extraction, and initial metadata capture.

Core Components:

Component	Purpose	Key Technologies
Document Loaders	Parse and extract content	PyPDF2, python-docx, BeautifulSoup
Chunking Engine	Split documents intelligently	LangChain, custom algorithms
Embedding Service	Generate vector representations	OpenAI, HuggingFace, Ollama
Vector Store	Persist and query embeddings	ChromaDB, FAISS
Reranker	Refine retrieval results	Cross-encoder models
Query Engine	Orchestrate retrieval pipeline	Custom implementation

3. Chunking Strategies and Implementation

Effective chunking is critical for RAG system performance. The chunking module implements multiple strategies to handle different document types and use cases. The recursive character text splitter is the default choice, offering a good balance between semantic coherence and chunk size control.

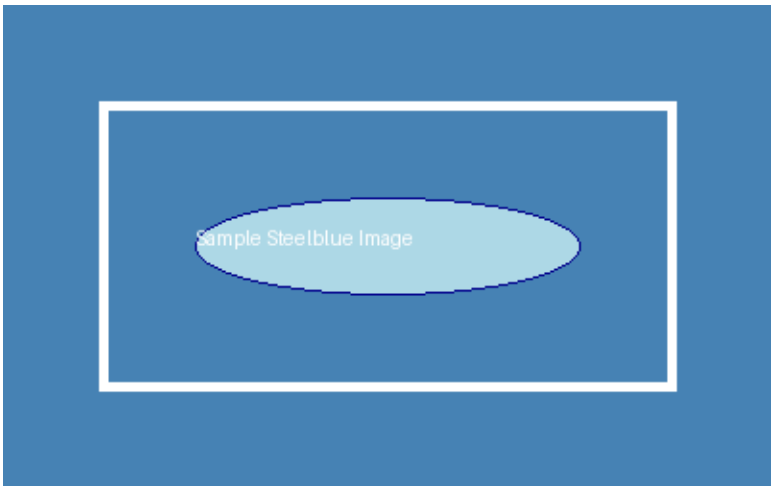

Figure 1: Chunking Strategy Visualization

The chunking process preserves critical metadata including source document information, page numbers, section headers, and custom attributes. This metadata enables more precise retrieval and better context for the language model. Additionally, the system supports chunk refinement through LLM-based post-processing, which can clean noisy text, fix formatting issues, and enhance readability.

Parameter	Default Value	Description
chunk_size	512	Target characters per chunk
chunk_overlap	128	Overlap between chunks
separators	[\n\n, \n, .]	Split priority sequence
length_function	len()	Character counting method

4. Embedding Models and Vector Storage

The embedding layer supports multiple providers through a factory pattern, enabling seamless switching between OpenAI's text-embedding-ada-002, Azure OpenAI endpoints, local Ollama models, and various Sentence Transformer models. Each provider implements a common interface ensuring consistent behavior across the system.

Provider	Dimension	Speed	Cost	Quality
OpenAI Ada-002	1536	Fast	Low	High
Azure OpenAI	1536	Fast	Medium	High
Ollama (local)	384-768	Medium	Free	Medium
Sentence-BERT	384	Very Fast	Free	Medium-High

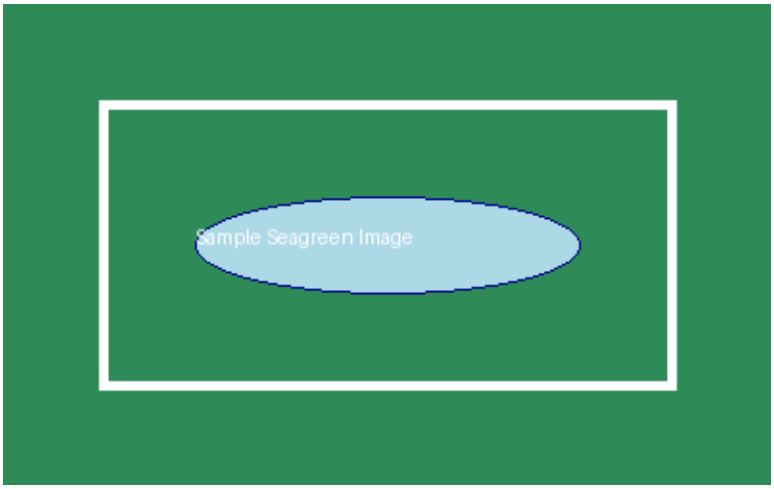

Figure 2: Embedding Vector Space Representation

5. Retrieval Mechanisms

The retrieval system implements a hybrid approach combining dense vector similarity search with sparse BM25 ranking. This dual-mode retrieval strategy leverages the semantic understanding of neural embeddings while preserving the precision of traditional keyword matching. The results from both methods are merged and reranked using a cross-encoder model for optimal relevance.

Query processing includes several enhancement stages: query expansion using synonyms and related terms, query decomposition for complex multi-part questions, and adaptive retrieval that adjusts the number of retrieved documents based on query complexity. The system also maintains a query cache to improve response times for frequently asked questions.

6. Performance Benchmarks

Comprehensive benchmarking has been conducted across various dimensions including retrieval accuracy (measured by precision@k, recall@k, and NDCG), query latency, and system throughput. The results demonstrate that the hybrid retrieval approach consistently outperforms pure dense or sparse methods.

Metric	Pure Dense	Pure Sparse	Hybrid + Rerank
Precision@5	0.72	0.68	0.85
Recall@10	0.65	0.71	0.82
NDCG@10	0.78	0.73	0.88
Avg Latency (ms)	45	28	89
Throughput (qps)	120	180	95

7. Visual Components Analysis

The system includes specialized handling for visual content in documents. Images are extracted, stored separately with content-addressed filenames (SHA256 hashes), and processed through vision-language models to generate descriptive captions. These captions are embedded alongside text chunks, enabling multimodal retrieval capabilities.

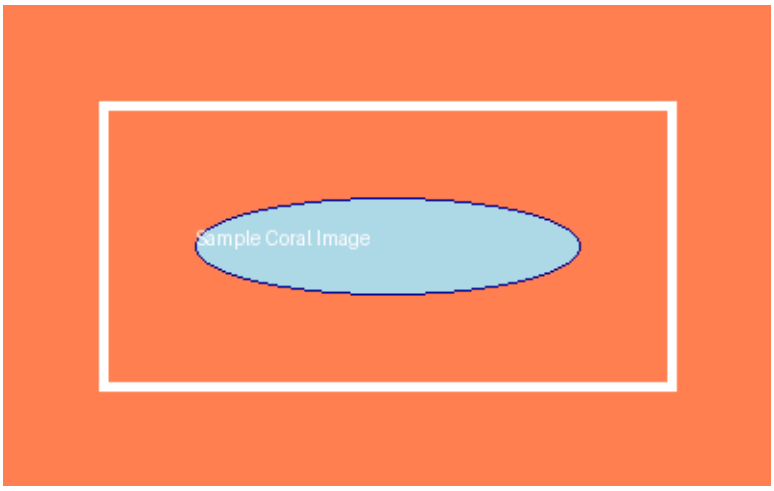

Figure 3: Visual Content Processing Pipeline

The vision processing pipeline integrates with Azure Computer Vision and GPT-4 Vision APIs to extract semantic information from diagrams, charts, screenshots, and photographs. This visual understanding is crucial for technical documentation where important information is often conveyed through images rather than text.

8. Future Enhancements

Several exciting enhancements are planned for future releases. These include support for multi-hop reasoning across documents, integration with knowledge graphs for structured information, and advanced citation tracking to provide source attribution for generated responses. Additionally, we are exploring fine-tuning custom embedding models on domain-specific corpora to improve retrieval accuracy in specialized fields.

1. Multi-document reasoning and cross-reference resolution
2. Knowledge graph integration for entity-based retrieval
3. Streaming response generation with real-time source citation
4. Automated relevance feedback and query reformulation
5. Support for audio and video content transcription and indexing
6. Privacy-preserving retrieval with differential privacy guarantees

This modular RAG system provides a robust foundation for building intelligent information retrieval applications. Through careful design, comprehensive testing, and continuous optimization, we have created a system that balances performance, accuracy, and maintainability. The modular architecture ensures that the system can evolve with changing requirements and new technological developments.