# Phase 3 – Retrieval-Augmented Generation (RAG)

## 🔹 RAG Architecture

1. Ingestion Layer  
- Document upload, parsing, and chunking (PDF, DOCX, Markdown, SQL text).  
- Clean text storage + chunking by semantic boundaries.  
  
2. Embedding & Storage Layer  
- Generate vector embeddings (OpenAI or HuggingFace).  
- Store in Postgres + pgvector or Azure Cognitive Search.  
- Maintain metadata in SQL (docId, chunkId, tags, owner, version).  
  
3. Retrieval Layer  
- Semantic + hybrid search over embeddings.  
- Ranking, deduplication, filters.  
  
4. Augmentation + Generation Layer  
- Retrieved chunks + user query → LLM.  
- Compare baseline LLM vs RAG-enhanced responses.  
- Debug mode: show retrieved chunks in Admin Panel.

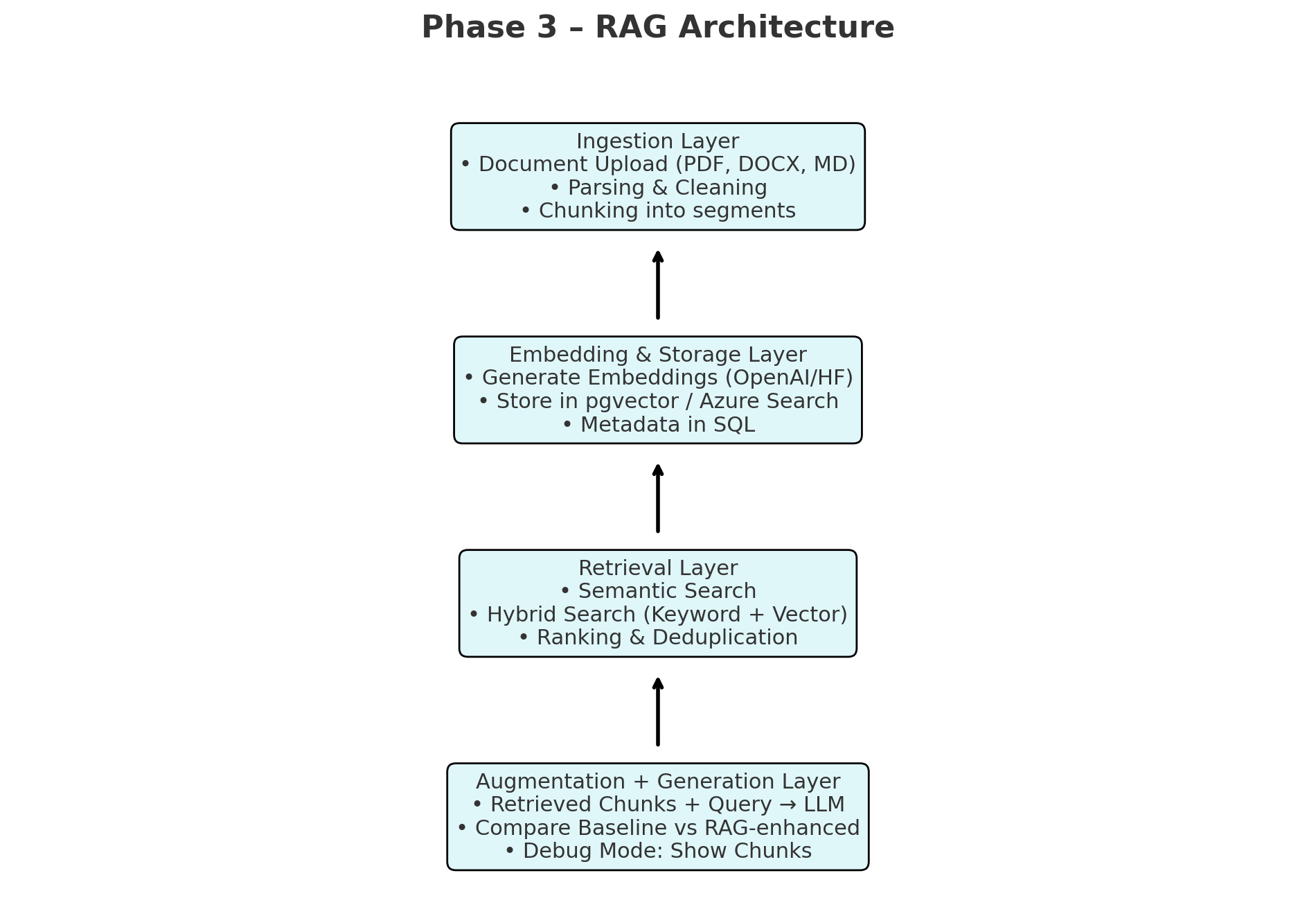
## 🔄 Phase 3 Sub-Phases

Phase 3.1 – Document Ingestion & Chunking ✅ (Completed)

- Admin panel document upload (PDF, DOCX, TXT, Markdown).  
- Parsing & validation with configurable AllowedTypes + MaxFileSize.  
- Store text + chunks in SQL, original files in wwwroot/uploads.  
- DTO-based service layer, business logic in DocumentService.  
- AdminLTE UI: Upload, Edit (re-upload new file), Delete (with confirm modal), Details (preview).  
- Preview: PDF inline, TXT as chunks, DOCX fallback download.  
- Outcome: Robust ingestion pipeline, documents ready for embeddings.

## ⚡ Expected Outcomes

- RAG-enabled chatbot in .NET + AdminLTE project.  
- Admin panel for document + RAG debugging.  
- Comparison: baseline vs RAG-enhanced responses.  
- Scalable + production-ready retrieval pipeline.



### Updated Architecture (Phase 3.1)

