MODULE 1: RAG ARCHITECTURE OVERVIEW

📚 LEARNING OBJECTIVES

By the end of this module, you will be able to: - Define Retrieval Augmented Generation and explain its value - Identify the core components of a RAG system - Describe the typical RAG flow from ingestion to generation - Understand trade-offs in RAG system design - Map RAG concepts to the AI Accelerator framework

1. WHAT IS RAG?

DEFINITION

Retrieval Augmented Generation (RAG) is a technique that enhances Large Language Models (LLMs) by providing them with relevant external knowledge retrieved from a document corpus during the generation process.

```
Traditional LLM:            RAG-Enhanced LLM:

User Query                 User Query
    ↓                           ↓
   LLM  →  Answer           Retriever → Relevant Docs
                                 ↓
                             LLM + Context  →  Grounded Answer
```

THE PROBLEM RAG SOLVES

Challenge: LLMs have several limitations: 1. **Knowledge Cutoff**: Training data is frozen at a point in time 2. **Hallucinations**: Models may generate plausible but incorrect information 3. **Domain Specificity**: Lack of specialized or proprietary knowledge 4. **Attribution**: Difficulty in citing sources for generated content

Solution: RAG addresses these by: - Retrieving up-to-date, relevant information at query time - Grounding responses in actual source documents - Enabling domain-specific knowledge without retraining - Providing source attribution for transparency

REAL-WORLD USE CASES

2. CORE COMPONENTS OF RAG

2.1 DOCUMENT STORE / CORPUS

Purpose: The knowledge base containing documents to be searched

Characteristics: - **Format**: PDFs, HTML, markdown, plain text, structured data - **Scale**: From hundreds to millions of documents - **Domain**: General knowledge, domain-specific, or proprietary data

Example:

```
# Document structure
document = {
    "id": "doc_12345",
    "title": "Employee Handbook 2024",
    "text": "Company policies and procedures...",
    "metadata": {
        "author": "HR Department",
        "date": "2024-01-15",
        "category": "policies"
    }
}
```

2.2 CHUNKING / PREPROCESSING

Purpose: Split large documents into manageable, semantic units

Why Chunking Matters: - Embedding models have token limits (e.g., 512 tokens) - Smaller chunks = more precise retrieval - Balanced chunk size improves relevance

3. TYPICAL RAG FLOW

COMPLETE RAG PIPELINE

```
graph LR
    A[Raw Documents] -->|1. Extract| B[Text Content]
    B -->|2. Chunk| C[Document Chunks]
    C -->|3. Embed| D[Vectors]
    D -->|4. Index| E[Vector Store]

    F[User Query] -->|5. Embed| G[Query Vector]
    G -->|6. Search| E
    E -->|7. Retrieve| H[Relevant Chunks]
    H -->|8. Context| I[LLM]
    F -->|9. Question| I
    I -->|10. Generate| J[Answer]
```

PHASE 1: INGESTION (ONE-TIME SETUP)

Step 1: Extract

```python
# Extract text from various formats
from langchain.document_loaders import (
    PyPDFLoader,
    TextLoader,
    UnstructuredHTMLLoader
)

loader = PyPDFLoader("employee_handbook.pdf")
documents = loader.load()
```

Step 2: Chunk

4. TRADE-OFFS IN RAG DESIGN

4.1 LATENCY VS ACCURACY

Latency Factors: - Embedding model size - Vector store query time - Number of retrieved chunks (k) - LLM size and parameters

Optimization Strategies:

```
# Faster (lower latency)
embeddings = HuggingFaceEmbeddings("all-MiniLM-L6-v2")  # 384 dim
k = 3  # fewer chunks
llm = Ollama("llama2:7b")  # smaller model

# More accurate (higher latency)
embeddings = HuggingFaceEmbeddings("all-mpnet-base-v2")  # 768 dim
k = 10  # more chunks
llm = WatsonxLLM("ibm/granite-13b-chat-v2")  # larger model
```

4.2 EMBEDDING MODEL SELECTION

Considerations: - **Domain:** General vs specialized (legal, medical, etc.) - **Language:** Multilingual support - **Dimensions:** Higher = more nuance, slower - **License:** Open-source vs proprietary

Comparison: | Criteria | Small Model | Large Model | |———-|————-|————-| | Speed | ⚡⚡⚡ | ⚡ | | Accuracy | ⭐⭐ | ⭐⭐⭐ | | Memory | 💾 | 💾💾💾 | | Cost | $ | $$$ |

5. RAG IN THE AI ACCELERATOR

ACCELERATOR ARCHITECTURE

The AI Accelerator provides production-ready RAG components:

```
accelerator/
├── tools/
│   ├── chunk.py           # Document chunking
│   ├── extract.py         # Text extraction
│   └── embed_index.py     # Embedding & indexing
├── rag/
│   ├── retriever.py       # Retrieval logic
│   ├── pipeline.py        # End-to-end RAG
│   └── prompt.py          # Prompt templates
└── service/
    └── api.py             # FastAPI endpoints
```

COMPONENT DETAILS

1. INGESTION & INDEXING

`tools/chunk.py` - Document Chunking

```python
def chunk_documents(
    documents: List[Document],
    chunk_size: int = 1000,
    chunk_overlap: int = 200
) -> List[Document]:
    """Split documents into chunks"""
    pass
```

6. HOW DAY 2 LABS MAP TO RAG COMPONENTS

LAB PROGRESSION

```
Lab 2.1 (Local RAG)
    ↓
Learn: Basic RAG flow, Ollama integration, Chroma usage
Maps to: chunk.py, embed_index.py basics

Lab 2.2 (watsonx RAG)
    ↓
Learn: Enterprise RAG, Elasticsearch, watsonx.ai
Maps to: retriever.py, pipeline.py

Lab 2.3 (Twin Pipelines)
    ↓
Learn: Multi-backend orchestration, comparison
Maps to: Service layer design, api.py
```

PROGRESSIVE IMPLEMENTATION

Each lab builds toward a production RAG service:

1. **Lab 2.1**: Core concepts in notebook
2. **Lab 2.2**: Enterprise patterns
3. **Lab 2.3**: Service design
4. **Lab 2.4**: Quality assurance

By end of Day 2: - ✅ Understand all RAG components - ✅ Implement multiple RAG backends - ✅ Ready to integrate into accelerator - ✅ Evaluate and optimize RAG systems

SUMMARY

KEY TAKEAWAYS

1. **RAG = Retrieval + Generation**
   - Grounds LLM responses in external knowledge
   - Reduces hallucinations through factual context
2. **Core Pipeline**: Ingest → Index → Retrieve → Generate
3. **Critical Components**:
   - Chunking strategy
   - Embedding model
   - Vector store
   - Retrieval mechanism
   - LLM integration
4. **Trade-offs Matter**:
   - Latency vs accuracy
   - Index size vs freshness
   - Cost vs performance

ADDITIONAL RESOURCES

DOCUMENTATION

- LangChain RAG Guide
- watsonx.ai RAG Patterns
- Vector Database Comparison

PAPERS

- “Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks”
- “Dense Passage Retrieval for Open-Domain Question Answering”

TOOLS

- LangSmith - RAG debugging
- Weights & Biases - Experiment tracking

Theory Module Complete! ✅

Proceed to Lab 2.1 when ready.