<u>WEEK 7 — GENAI & MULTIMODAL RAG ENGINEERING</u>

(Text RAG + Image RAG + SQL Question Answering + Hybrid Retrieval + Local LLMs)

USE CASE: Enterprise Knowledge Intelligence System (Text + Images + SQL RAG)

Interns will build an enterprise-grade GenAI system that can:

- Answer questions from internal documents (PDFs, DOCX, TXT)
- Retrieve & reason over images (diagrams, charts, forms)
- Query structured data using natural language → SQL → result
- Run on **either** local open-source models **or** hosted LLM APIs (OpenAI / Claude / Gemini)
- Provide faithful, non-hallucinated responses with evaluation & monitoring

This simulates real enterprise systems used in:

- Banking (policy manuals + scanned KYC files + SQL transaction DB)
- Insurance (claims PDFs + damage images + SQL claim tables)
- Manufacturing (blueprints + manuals + operational DB)

DATASETS FOR THIS USE-CASE (WORKING + RELEVANT)

1) Tabular / Structured Data (CSV – Enterprise)

Enterprise CSV Samples (General Business Tables)

- Type: Tabular CSV files for business data (employees, products, transactions, etc.)
- Source: https://www.datablist.com/learn/csv/download-sample-csv-files

2) Document / Knowledge Data (Text-RAG)

Enterprise RAG Markdown Dataset

- Type: Markdown text documents — suitable for ingestion & retrieval tasks
- Source: https://www.kaggle.com/datasets/rrr3try/enterprise-rag-markdown

3) Graph / Structured Relationship Data

Graphs Dataset

- Type: Structured graph data (networks/relationships) — useful for entity graphs, KG-RAG
- Source: https://www.kaggle.com/datasets/sunedition/graphs-dataset

---

WEEK 7 OBJECTIVES

Interns will learn:
✔ Advanced Retrieval-Augmented Generation (RAG) architecture
✔ Local LLM setup & optimization **or** API-based LLM integration
✔ Hybrid retrieval (semantic + keyword + reranking + image)
✔ Multimodal embeddings for Image-RAG
✔ SQL-based question answering
✔ Prompt + context optimization
✔ Document chunking & metadata routing
✔ GenAI pipeline design (fully offline **or** “enterprise-controlled API mode”)

MODEL STACK (CHOOSE ONE PATH — OPEN-SOURCE LOCAL OR HOSTED APIS)

PATH A — OPEN-SOURCE LOCAL (Fully Offline)

LLMs (choose 1):

- Mistral-7B Instruct
- LLaMA-2 / LLaMA-3 local versions
- Qwen2
- Phi-3

Embeddings Models:

- BGE-small / BGE-base
- Instructor-XL
- GTE-base
- CLIP (for image embeddings)

Vector DB Options:

- FAISS
- Chroma
- Qdrant (local container)

OCR / Image Tools:

- Tesseract
- OpenCV
- BLIP / CLIP

SQL Database:

- PostgreSQL

- SQLite (recommended for interns)

PATH B — HOSTED LLMs (Keys + Model IDs)

Use this when interns don't have GPUs or you want "enterprise-grade APIs" while keeping the *same* RAG architecture, chunking, tracing, eval, and monitoring.

Option 1: OpenAI (API)

API Key env var: `OPENAI_API_KEY`
Common model choices: GPT-5.2 / GPT-5 mini / GPT-4.1 (and related variants). OpenAI Platform+2OpenAI+2

Option 2: Anthropic Claude (API)

API Key env var: `ANTHROPIC_API_KEY`
Common model choices: Claude 3.7 Sonnet (and other Claude family models listed in Claude docs). Anthropic+1

Option 3: Google Gemini (API)

API Key env var: `GOOGLE_API_KEY` *(Gemini API / AI Studio)*
Common model choices: Gemini 3 Pro / Gemini Flash family (per Gemini models docs). Google AI for Developers+1

> **Important:** Path B is *not* fully offline. For enterprise realism: add policy gates (PII redaction), logging controls, and strict "no training on customer data" vendor settings per your org's policy.

Minimal "Provider Switch" Config (keep your pipeline identical)

Create `config/model.yaml`:

- `provider: local | openai | anthropic | gemini`
- `model_name: <chosen model id>`
- `api_key_env: OPENAI_API_KEY | ANTHROPIC_API_KEY | GOOGLE_API_KEY`

Your code only changes in **one place**: `/generator/llm_client.py` (local loader vs API client). Everything else stays the same.

DAY 1 — LOCAL RAG SYSTEM + PIPELINE ARCHITECTURE

Learning Outcomes

- RAG architecture (Retriever → Generator)
- Local LLM loading and inference
- Embedding generation
- Document chunking strategies
- Semantic indexing (HNSW / IVF / Flat)

Mandatory Folder Structure

```
src/
  data/
    raw/
    cleaned/
    chunks/
  embeddings/
  vectorstore/
  retriever/
  generator/
  pipelines/
  prompts/
  models/
  evaluation/
  utils/
  config/
  logs/
```

Topics to Learn

- RAG architecture fundamentals
- Chunk size vs. token limits
- Overlap strategy
- Embedding pipelines
- Metadata tagging
- Vector index structures

Exercise

Build a local ingestion & chunking pipeline:
Pipeline must:

- Load PDFs / TXT / CSV / DOCX
- Clean text → split into 500–800 token chunks
- Add metadata (source, page number, tags)
- Generate local embeddings
- Store vectors in FAISS or Qdrant
- Build retriever module

✔ Documents loaded
✔ Chunks created
✔ Embeddings generated
✔ Vector DB initialized

Deliverables

- `/pipelines/ingest.py`
- `/embeddings/embedder.py`

- `/vectorstore/index.faiss`
- `/retriever/query_engine.py`
- `RAG-ARCHITECTURE.md`

DAY 2 — ADVANCED RETRIEVAL + CONTEXT ENGINEERING

Learning Outcomes

- Improve retrieval accuracy
- Build hybrid retrieval (semantic + keyword + reranking)
- Context ranking strategies
- Reduce hallucination

Topics

- Hybrid search: BM25 + embeddings
- Reranking (cross-encoder / cosine)
- Max Marginal Relevance (MMR)
- Chunk deduplication
- Optimizing context window for LLMs

Exercise

Build an advanced retriever that supports:

```
query = "Explain how credit underwriting works"
```

```
top_k = 5
filters = { "year": "2024", "type": "policy" }
```

Add:

- Keyword fallback
- Reranking
- Deduplication
- Traceable context sources

✔ Higher precision
✔ Lower hallucination
✔ Fully traceable context

Deliverables

- `/retriever/hybrid_retriever.py`
- `/retriever/reranker.py`
- `/pipelines/context_builder.py`
- `RETRIEVAL-STRATEGIES.md`

DAY 3 — IMAGE-RAG (MULTIMODAL RAG)

Learning Outcomes

- Handle images inside RAG
- Generate & store vision embeddings
- OCR extraction + captioning

- Image similarity search

Topics

- CLIP embeddings (image + text)
- OCR extraction using Tesseract
- Caption generation using BLIP
- Multimodal vector DB design

Exercise

Build an Image RAG pipeline:
Supports ingestion of:

- PNG, JPG, scanned PDFs, forms, diagrams

System generates:

- OCR Text
- CLIP embeddings
- Captions (BLIP)
- Multimodal vector index

Query modes:

- Text → Image
- Image → Image
- Image → Text Answer

Example:
User uploads an engineering diagram → System retrieves related diagrams + explanations.

Deliverables

- `/pipelines/image_ingest.py`
- `/embeddings/clip_embedder.py`
- `/retriever/image_search.py`
- `MULTIMODAL-RAG.md`

DAY 4 — SQL QUESTION ANSWERING SYSTEM (Text → SQL → Answer)

Learning Outcomes

- Convert natural language queries into SQL
- Schema-aware reasoning
- SQL query correction / validation
- Injection-safe execution

Topics

- Schema extraction
- Prompting patterns for SQL generation
- Query validation
- Error correction
- Summarizing result tables

Exercise

Build a SQL-QA Engine:
User: “Show total sales by artist for 2023.”

System should:

- Generate SQL using LLM
- Validate SQL
- Execute on SQLite/PostgreSQL
- Summarize the results

Features:
✔ Auto schema loader
✔ Query validator
✔ Safe executor
✔ Result summarizer

Deliverables

- `/pipelines/sql_pipeline.py`
- `/generator/sql_generator.py`
- `/utils/schema_loader.py`
- `SQL-QA-DOC.md`

DAY 5 — ADVANCED RAG + MEMORY + EVALUATION (CAPSTONE)

Learning Outcomes

- Add conversational memory
- Self-reflection + refinement loops
- Hallucination detection
- Faithfulness scoring
- Production-ready API structure

Topics

- Memory storage (Vector + Redis + Local File)
- Self-critique for improving answers
- Context match score
- Faithfulness scoring
- Human feedback logging

Exercise (Capstone)

Build a complete system with endpoints:

- `/ask`
- `/ask-image`
- `/ask-sql`

Add:
✔ Memory for last 5 messages
✔ Refinement loop
✔ Hallucination detection
✔ Confidence score
✔ Logging + debugging traces
✔ Streamlit UI or CLI

Deliverables

- `/deployment/app.py`
- `/evaluation/rag_eval.py`
- `/memory/memory_store.py`
- `CHAT-LOGS.json`
- `DEPLOYMENT-NOTES.md`

WEEK 7 COMPLETION REQUIREMENTS

Skill Area	Requirement
RAG System	End-to-end functioning
Advanced Retrieval	Hybrid + reranking
Image RAG	CLIP + OCR + multimodal vectors
SQL QA	Natural language → SQL → Answer
Memory	Context retention
Evaluation	Faithfulness & context matching
Local Models / API Models	Works with chosen stack
Documentation	All systems documented

EXPECTED OUTCOME

After this week, interns will be able to:
✔ Build enterprise-grade GenAI systems
✔ Handle documents + images + SQL databases
✔ Run LLMs locally **or** via OpenAI/Claude/Gemini keys
✔ Reduce hallucination by over 70%
✔ Build multimodal & hybrid RAG pipelines
✔ Prepare for multi-agent GenAI scaling