Week 9 – RAG Lecture Notes

RAG Overview

Retrieval-Augmented Generation (RAG) enhances LLMs by fetching external knowledge before generating responses.

This allows LLMs to stay up-to-date without retraining on new information.

RAG is especially useful for knowledge-intensive tasks such as fact-based QA, legal reasoning, or domain-specific support.

Information Retrieval Foundations

Information Retrieval (IR) focuses on finding relevant documents from a large corpus based on a user query.

Three components of IR:

• Embedding: converting text into numerical vectors.

• Similarity modelling: determining how close two vectors are.

• Indexing: retrieving efficiently from large databases.

Embedding Methods

Discrete embeddings: based on lexical frequency (TF, TF-IDF, binary representation). Good for keyword-heavy tasks but limited semantically.

Continuous embeddings: learned numerical representations that capture semantic meaning (e.g., BERT, sentence transformers). Better for modern retrieval.

Similarity Models

Cosine similarity: measures the angle between vectors; widely used for dense embeddings.

Jaccard similarity: measures overlap between sets; used in discrete or binary features.

BM25: a keyword-based scoring method that considers term frequency and rarity; strong baseline for text retrieval.

Indexing

Indexing ensures fast retrieval even in billions of documents.

Sparse indexing (inverted index) stores which docs contain each term—efficient for keyword search.

Dense vector indexing uses clustering (e.g., FAISS) to avoid scanning the entire vector space.

Naive RAG Pipeline

Step 1: Indexing — split documents into chunks and embed them.

Step 2: Retrieval — retrieve top-k relevant chunks using similarity search.

Step 3: Generation — feed retrieved chunks into LLM along with query to produce answer.

Advanced RAG Enhancements

Sliding window: overlapping chunks avoid losing information at boundaries.

Metadata: adds structure (titles, sections, timestamps) to improve retrieval accuracy.

Fine-grained segmentation: retrieves smaller units (paragraphs, sentences) for precision.

Retrieval Routes

Instead of searching everything flatly, RAG can choose different retrieval paths depending on query type.

Examples include: document-first retrieval, summary-first retrieval, or object-level retrieval (tables, figures).

This improves accuracy, efficiency, and alignment with query intent.

Summarization in Retrieval

Summaries serve as coarse-grained retrieval units.

The system can first retrieve summaries, then explore associated sections or chunks.

Useful for long documents, reports, or books.

Query Rewriting

Queries may be vague, incomplete, or lack keywords.

Rewriting makes queries clearer: adds missing context, expands abbreviations, or makes structure explicit.

This leads to significantly better retrieval performance.

Confidence Judgment

Before generating final output, RAG checks confidence in both retrieval and generation.

Retrieval confidence: similarity scores, metadata consistency, redundancy in evidence.

Generation confidence: LLM self-checking, groundedness, contradiction detection.

Post-Retrieval: Reordering & Filtering

Reordering: prioritizing most relevant evidence early because LLMs weigh earlier content more.

Filtering: removing irrelevant or low-quality chunks to prevent hallucinations.

These steps make the final answer more accurate and trustworthy.

RAG Paradigms

Rewrite → Retrieve → Read: rewriting improves retrieval precision.

Retrieve → Read → Generate: classical RAG approach.

DSP (Demonstrate–Search–Predict): uses demonstrations before retrieval.

Key Problems in RAG

How to retrieve: chunk-level, entity-level, or KG-level.

When to retrieve: once, adaptively, or periodically during generation.

How to use retrieved information: merging, filtering, re-ranking.

Key RAG Technologies

Chunk optimization: sentence-level embeddings, sliding windows, multi-stage retrieval.

Structured corpus: metadata to preserve section relationships and filter irrelevant docs.

KG retrieval: retrieving subgraphs via extracted entities.

Query optimization: rewriting, clarifying queries, structured search prompts.

Embedding optimization: choose the best embedding model; fine-tune for domain.

Fine-tuning retriever: aligning retriever with LLM's preferred evidence.