Project Report: Local RAG System for Document Search & Summarization

1. Objective and Scope:

The objective of this project is to build a system that can **search** a sizable text corpus and **summarize** the most relevant results using a Large Language Model (LLM).

The system was implemented to satisfy the assignment tasks:

- **Data preparation:** choose a sizable corpus, clean it, and make it usable for search/summarization.
- **Document search:** ingest user queries, retrieve relevant excerpts using a combination of **traditional IR** and **LLM embeddings**, and return **Top-N** results.
- **Summarization:** summarize retrieved excerpts with an LLM, ensuring coherence and allowing the user to choose the summary length.
- **Evaluation:** build a test set, generate queries, measure retrieval accuracy, and evaluate summaries using automated metrics (e.g., ROUGE).
- **Interface (bonus):** implement a user-friendly UI with auto-suggestions, pagination, and adjustable summary length.

2. System Overview (Architecture):

High-level flow:

1. **Offline corpus build:**

- Load corpus (Wikipedia)
- Clean text
- Chunk documents
- Save to corpus.json

2. **Retrieval-Augmented Generation (RAG) runtime:**

- Load chunks → LangChain Documents
- Build:
  - Dense index (FAISS + embeddings)
  - Sparse index (BM25)
- Hybrid retrieve Top-K
- Summarize retrieved context with local LLM

3. **User interface:**

- Streamlit app for query input, summary length selection, suggestions, and paginated results

4. **Evaluation:**

- Create a test subset

- Auto-generate queries per document
- Compute Recall@K and ROUGE

3. <u>Data Preparation:</u>

3.1 <u>Corpus Choice:</u>

- **Corpus:** wikimedia/wikipedia (Simple English snapshot 20231101.simple)
- **Subset used for indexing:** NUM_DOCUMENTS = 1000 articles (development-sized subset)

This satisfies the requirement to choose a “sizable corpus” and prepare it for retrieval and summarization.

3.2 <u>Cleaning:</u>

Implemented clean_text() to reduce noise and stabilize chunking/embedding:

- Removed bracket citations like [1]
- Normalized whitespace and trimmed ends

Why it matters: Noisy tokens (citations, irregular whitespace) can degrade BM25 scoring, embedding quality, and summary grounding.

3.3 <u>Chunking Strategy:</u>

Used RecursiveCharacterTextSplitter with:

- CHUNK_SIZE = 500
- CHUNK_OVERLAP = 50
- separators: paragraph → line → sentence → word → char fallback

<u>Reasoning:</u>

- Retrieval works best when documents are split into semantically meaningful, bite-sized chunks.
- Overlap reduces boundary loss (important facts split between chunks).

3.4 <u>Output Format:</u>

Saved all chunks into a single JSON corpus file:

- data/processed/corpus.json

Each entry includes:

- id (chunk id)
- doc_id (original Wikipedia id)
- title, url
- content
- chunk_index

This structured output is designed to support:

- retrieval (BM25/FAISS)
- UI traceability (title/source id)
- evaluation grouping (doc_id, chunk_index)

4. <u>Document Search Methodology:</u>

The assignment requires combining **traditional IR** and **LLM embeddings**, and returning the **Top-N** results.

4.1 <u>Dense Retrieval (Semantic):</u>

- **Embeddings:** HuggingFaceEmbeddings(model_name="all-MiniLM-L6-v2")
- **Vector index:** FAISS (FAISS.from_documents(...))
- **Retriever:** as_retriever(k=TOP_K)

<u>Benefits:</u>

- Finds semantically similar content even with different wording.
- Handles paraphrases better than keyword search.

4.2 <u>Sparse Retrieval (Keyword):</u>

- **Retriever:** BM25Retriever.from_documents(...)
- k = TOP_K

<u>Benefits:</u>

- Strong for exact-match entities (names, dates, jargon).
- Often outperforms dense retrieval on rare keywords.

4.3 <u>Hybrid Search (Ensemble Retriever):</u>

- **EnsembleRetriever** combining:
  - dense retriever
  - sparse retriever
- weights: [0.5, 0.5]

Why hybrid matters: It aligns with the requirement to combine IR + embeddings for better accuracy.

Gen AI Engineer - RAG

4.4 <u>Top-K Output:</u>

- TOP_K = 5
- Returned to user as the “most relevant excerpts” for the query.

5. Document Summarization Methodology:

The assignment requires summarizing retrieved results with an LLM, ensuring coherence and supporting adjustable length.

5.1 Local LLM (Offline):

- **LLM runtime:** Ollama
- **Model:** llama3.1:8b
- Temperature set to 0 for more consistent, less "creative" outputs.

This also follows the suggestion to consider resource constraints (LLMs can be resource-intensive).

5.2 Prompting and Grounding:

The summarizer constructs a context block:

- concatenates retrieved chunks
- labels each chunk with its title: Source (title): chunk_text

Prompt includes constraints:

- "Based ONLY on the provided context"
- "Do not introduce outside information"

Goal: reduce hallucinations and keep answers grounded in retrieved text.

5.3 Adjustable Summary Length:

Implemented length_guidelines mapping:

- short: 2–3 sentences
- medium: detailed paragraph
- long: report with bullets/sections

This directly satisfies the requirement that users can specify summary length.

6. Interface (Bonus): Streamlit App:

The assignment bonus suggests a UI with:

- query input
- auto-suggestions
- pagination
- adjustable summary length

6.1 Features Implemented:

- **Search bar + Search button**
- **Suggestion pills** (quick demo queries)

- **Summary length slider** (short/medium/long)
- **Paginated results** (3 results per page)
- **Expandable result cards** showing chunk text and metadata
- **Clear history** button (resets session state)

6.2 <u>Performance Optimization:</u>

Used @st.cache_resource to cache the engine so that Streamlit reruns do not rebuild:

- embeddings loading
- FAISS index
- BM25 index

This improves responsiveness and supports “efficiency” evaluation.

7. <u>Evaluation Methodology:</u>

The assignment requires:

- create a test subset
- generate a query for each document
- measure retrieval accuracy
- evaluate summary quality with automated metrics like ROUGE

7.1 <u>Test Set Construction:</u>

- Used the processed corpus and grouped chunks by doc_id.
- Filtered to documents with at least 3 chunks (enough content for meaningful evaluation).
- Sampled TEST_SET_SIZE = 20 parent documents with RANDOM_SEED = 42.

7.2 <u>Query Generation (Synthetic):</u>

For each test document:

- chose a chunk from the **middle 50%** of the article (harder than intro)
- used the LLM to generate a realistic search query without mentioning the title

This tests retrieval beyond “easy intro keyword” cases.

7.3 <u>Retrieval Metric: Recall@K</u>

- Retrieved results using the hybrid retriever
- Checked **Recall@5** by verifying whether the correct doc_id appears among Top-5 retrieved chunks.

7.4 <u>Summarization Metric: ROUGE</u>

Computed ROUGE against two references:

1. **Lead chunk reference (Intro):** measures “topic coverage”
2. **Source chunk reference (the chunk used to generate the query):** measures “answer accuracy”

Stored per-case logs in evaluation_results.csv for auditability.

8. Evaluation Results:

Metric	**Result**
Recall@5 (Hit Rate)	**100.0%**
ROUGE-1 vs Intro (Topic Coverage)	**0.3077**
ROUGE-1 vs Source Chunk (Answer Accuracy)	**0.4710**

Interpretation:

- Retrieval is very strong on this corpus subset (Top-5 always included at least one chunk from the correct document).
- Summaries align better with the **specific source chunk** than the intro, which is expected because queries are generated from mid-article content.

9. Challenges Faced and Solutions:

9.1 ID consistency (doc_id type mismatch):

Problem: doc_id can appear as int/string depending on source and pipeline.
Solution: forced string casting during grouping and comparison.

9.2 “Intro bias” in evaluation:

Problem: queries generated from intros often become too easy.
Solution: generate queries from the middle of articles to make retrieval more realistic.

9.3 Strict Recall@K testing:

Problem: retrievers may return more than K items depending on implementation.
Solution: explicitly slice retrieved results to [:TOP_K_CHECK] to enforce Recall@K.

9.4 Streamlit rerun cost:

Problem: Streamlit reruns re-execute scripts; rebuilding FAISS/BM25 each rerun is slow.
Solution: cached engine using st.cache_resource.

10. Scalability and Efficiency Considerations:

The assignment emphasizes scalability and efficiency.

Current design choices supporting this:

- **Streaming dataset loading** (doesn’t require loading full Wikipedia into memory).
- **Chunking + indexing** enables scalable retrieval rather than LLM reading full documents.

- **FAISS** provides fast approximate nearest-neighbor search.
- **Caching in UI** prevents repeated expensive initialization.

Potential improvements for larger corpora:

- Persist FAISS index to disk (avoid rebuilding each run).
- Batch embedding computation and/or GPU acceleration for preprocessing.
- Add a re-ranker (cross-encoder) on Top-K to boost precision.

11. <u>Limitations:</u>

- Evaluation uses **synthetic queries** generated from document text; this can inflate retrieval performance compared to truly human-written queries.
- ROUGE is a lexical overlap metric; it may underrate correct summaries that paraphrase well.
- Current pipeline summarizes only retrieved chunks; if retrieval misses critical context, summary quality degrades.

12. <u>Future Work:</u>

- Add human evaluation alongside ROUGE (explicitly suggested).
- Add citation-style answers (quote snippet + source title/url).
- Improve chunking (sentence-aware chunking; dynamic chunk sizes).
- Add query rewriting for better recall.
- Explore domain adaptation / fine-tuning as hinted.

13. <u>Conclusion:</u>

This project delivers a complete local RAG pipeline that:

- prepares and chunks a sizable corpus
- retrieves top-N results using hybrid BM25 + embeddings
- summarizes results with adjustable length
- evaluates retrieval and summarization with Recall@K and ROUGE
- provides a UI with suggestions, pagination, and length control