Individual Final Report

Project Title: AWS CloudGuide - Your Cloud Architecture Assistant

Student Name: Mohammed Ismail Sarfaraz Shaik

1. Introduction

The primary objective of this project was to engineer an Advanced Hybrid Retrieval-Augmented Generation (RAG) system capable of functioning as an autonomous AWS Technical Architect. The system was designed to ingest a massive corpus of technical documentation (over 100,000 pages) and utilize the Qwen-2.5-7B-Instruct Large Language Model (LLM) to deliver high-precision, citation-backed technical support.

Scope of Work & Project Architecture: The project architecture was decoupled into three functional layers: the Data & Indexing Layer, the Backend Retrieval Logic, and the Frontend Evaluation Interface.

My individual responsibility was the complete design and implementation of the **Data & Indexing Layer**. This foundational component required:

1. **Data Acquisition & Cleaning:** Sourcing raw PDF data and sanitizing it for NLP tasks.
2. **Vectorization Architecture:** Managing GPU resources to generate high-dimensional embeddings.
3. **Hybrid Index Construction:** Creating synchronized Dense (Vector) and Sparse (Lexical) indices to support advanced retrieval strategies.

2. Description of Individual Work

My contribution focused on establishing the "Knowledge Base" infrastructure. In a RAG system, the quality of generation is strictly bound by the quality of retrieval. To ensure the LLM could access both conceptual and specific technical data, I architected a **Hybrid Search Index** combining two distinct information retrieval algorithms:

1. **Semantic Search (Dense Retrieval):** I implemented the **BAAI/bge-m3** embedding model to project text chunks into a 1024-dimensional vector space. This layer handles conceptual queries (e.g., "How do I optimize database throughput?") by utilizing Cosine Similarity to find vectors with semantic proximity, rather than mere word overlap.
2. **Lexical Search (Sparse Retrieval):** I implemented the **Okapi BM25 (Best Matching 25)** algorithm. This probabilistic framework ranks documents based on exact keyword frequency and inverse document frequency. This layer is critical for technical domains, ensuring that specific error codes, instance types (e.g., "t3. micro"), or rigid configuration parameters are retrieved accurately.

The integration of these two indices provides the downstream retrieval engine with the necessary data to perform Reciprocal Rank Fusion (RRF).

3. Detailed Description of Work

My implementation strategy was executed in three phases: Preprocessing, Resource-Optimized Ingestion, and Index Synchronization.

3.1 Data Ingestion and Preprocessing Pipeline

I utilized the semihk1/aws-public-pdf-chunked-dataset as the raw source, comprising parsed text from official AWS User Guides (2025 versions). To prevent "garbage-in, garbage-out" scenarios, I engineered a strict filtration pipeline:

- **Noise Reduction:** I developed regex-based filters to identify and excise "Table of Contents" artifacts (e.g., sequences of dots) and legal boilerplate (e.g., "Copyright," "All rights reserved"). This maximization of information density ensures that vectors represent technical content rather than document metadata.
- **Granularity Validation:** I enforced a minimum character threshold (100 chars), discarding micro-chunks that lacked sufficient context to form a meaningful semantic vector.

3.2 Chunking Strategy and GPU Architecture

Handling the scale of AWS documentation required a robust chunking and embedding strategy optimized for hardware constraints.

- **Recursive Context Windows:** I deployed a RecursiveCharacterTextSplitter with a window size of 1,000 characters and a 200-character overlap. This overlap is crucial for preserving semantic continuity across split boundaries, ensuring that multi-sentence technical explanations are not severed.
- **Batched GPU Memory Management:** The primary engineering challenge was generating embeddings on an NVIDIA T4 GPU (16GB VRAM) without causing Out-Of-Memory (OOM) errors. I implemented a batched ingestion loop that processes the corpus in controlled sets of 1,000 documents. This iterative approach allowed for the stable generation of over 800,000 embeddings, decoupling the dataset size from the GPU's physical memory limits.

3.3 Hybrid Index Synchronization

To enable Hybrid RAG, the Dense and Sparse indices must remain perfectly aligned. I developed a serialization script using Python’s pickle module to store the BM25 index structure alongside the ChromaDB vector store. This ensures that a document retrieved by ID doc_123 in the vector

store corresponds exactly to the same text segment in the BM25 index, enabling valid result fusion downstream.

4. Results

The success of the Data & Indexing Layer is measured by the scale, efficiency, and retrieval accuracy of the resulting database.

Quantitative Performance Metrics:

- **Corpus Scale:** 741 unique AWS User Guides ingested.
- **Index Volume:** ~813,000 searchable vector embeddings.
- **Storage Footprint:** ~8.5 GB (ChromaDB Vector Store) + ~1.2 GB (BM25 Index).
- **Throughput:** Achieved stabilized ingestion with retrieval latencies averaging <200ms.

Qualitative Validation: Tests confirmed the efficacy of the hybrid approach:

- **Precision (BM25):** Queries containing specific configuration flags (e.g., "S3 bucket policy syntax") successfully triggered the lexical index, retrieving exact syntax guides.
- **Concept (Vector):** Abstract queries (e.g., "Strategies for cost reduction") correctly utilized the bge-m3 vectors to surface architectural patterns from the AWS Well-Architected Framework, proving semantic preservation.

5. Summary and Conclusions

I successfully engineered a production-grade ingestion pipeline capable of supporting a large-scale Hybrid RAG application. The system effectively balances the semantic understanding of modern transformers with the precision of traditional information retrieval.

Key Technical Learnings:

1. **Hardware-Aware Programming:** Large-scale NLP tasks require explicit memory management (batching and garbage collection) rather than relying on default library behaviors.
2. **The Necessity of Hybrid Search:** In technical domains, semantic search alone is insufficient for disambiguating specific acronyms (e.g., "EBS" vs "EFS"); the inclusion of a lightweight BM25 index significantly boosts hit rates for entity-specific queries.
3. **Data Hygiene:** The pre-filtering of non-semantic text (headers/footers) had a measurable impact on retrieval quality, preventing the LLM from retrieving irrelevant page numbers or copyright dates.

Future Improvements: Future iterations would benefit from decoupling the ingestion process using an asynchronous queue (e.g., Celery or AWS SQS). This would allow for real-time

document updates without necessitating the blocking, linear re-indexing process currently implemented.

6. Code Percentage Calculation

The implementation relied on the LangChain and SentenceTransformers libraries for core abstractions, while the logic for data pipeline management, memory optimization, and hybrid synchronization was custom written.

- **Code Source:** LangChain documentation (Text Splitters, Vector Store interfaces) and Hugging Face (Model loading).
- **My Contribution:**
    - Custom data cleaning and regex filtration logic.
    - The batched processing loop for GPU memory management.
    - BM25 serialization and index synchronization logic.
    - Directory traversal and file handling structure.
    - Contributed to developing the working interface using Streamlit

Calculation:

- **Total Lines of Code:** ~305 lines.
- **Lines adapted from libraries/tutorials:** ~120 lines (Imports, class initializations, boilerplates).
- **Lines written from scratch:** ~185 lines (Processing logic, loops, custom handlers).
- **Percentage of Own Code: 60.6%**

7. References

1. Lewis, P., et al. (2020). *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*. arXiv:2005.11401.
2. Xiao, S., et al. (2023). *BGE-M3: The Mother of All Embeddings*. Beijing Academy of Artificial Intelligence.
3. Robertson, S., & Zaragoza, H. (2009). *The Probabilistic Relevance Framework: BM25 and Beyond*. Foundations and Trends in Information Retrieval.
4. LangChain AI. (2024). *Vector Stores and Retrievers Documentation*.
5. Amazon Web Services. (2025). *AWS Documentation & User Guides*. https://docs.aws.amazon.com/