AWS CloudGuide - Your Cloud Architecture Assistant

Maddala Harshith (G31253960)
Data Science Program
Columbian College of Arts & Science

INTRODUCTION

This project focuses on building an AWS assistant which is based on **Retrieval-Augmented Generation** (RAG) system, and is designed to support AWS cloud architecture decision-making. This chatbot has a combination of a custom backend which is capable of retrieving relevant AWS documentation, whitepapers, and best-practice guidelines, with a large language model which generates a context aware response, together displayed on a streamlined **Streamlit front-end**. The goal here is simple, we provide fast, simple and source-based guidance for users such as students, engineers, and practitioners working with AWS services. By integrating the retrieval with generation, we reduce hallucinations, ground each answer with verified AWS material, and deliver a reliable cloud-architecture helper.

Keywords: AWS assistant, Retrieval-Augmented Generation (RAG) system, cloud architecture, hybrid retrieval + LLM, Streamlit interface.

This individual report outlines the following sections:
1) Background of the project
2) Description of my individual work
3) Results
4) Summary
5) Percentage of code
6) References

I. BACKGROUND OF THE PROJECT

The AWS Architect Assistant is based on Retrieval-Augmented Generation (RAG) pipeline that has three core components: data ingestion, embedding + keyword indexing, and an accurate hybrid retrieval backend powered by a large language model (LLM).

In the ingestion stage we download the AWS documentation dataset from **Hugging face** and we apply quality filtering to remove unwanted data such as table of components pages, copyright text and extremely short or low-value chunks. The cleaned output is then stored as JSON lines, which helps by creating a uniform structure for downstream processing. Next comes the embedding phase and we start converting the processed documents into searchable representations. Large text segments are split into smaller (with necessary overlapping) and context preserving chunks. This is now encoded into vector embeddings using **BAAI/bge-m3**, an excellent retrieval method. In parallel, a **BM25** keyword index is built to capture exact term matches. These two retrieval methods

are combined through **Reciprocal Rank Fusion (RRF)**, which improves stability and ensures that both semantic similarity and keyword accuracy influence the final ranking for enhanced retrieval accuracy.

The backend implements an flawless workflow: user queries are rewritten for clarity, relevant documentation is extracted through hybrid search, and the final answer is generated using **Qwen2.5-7B-Instruct**, which is obligated to respond only with information found in the retrieved context. This design helps in reducing hallucinations, preserves AWS terminology, and yields reliable technical explanations grounded firmly in official AWS documentation

---

II. DESCRIPTION OF MY INDIVIDUAL WORK

My primary contribution to this project is focused on the evaluation **metrics** and the development of streamlit powered interface that showcases the overall AWS Architect Assistant to users. After the RAG pipeline was established, I came up with a methodology to measure the effectiveness of the system retrieval and documentation-based answer generation, that is, **Cosine Similarity**, a standard metric widely used in retrieval-augmented systems. The objective here is simple. We need to check how closely the model's generated answer measures with an ground truth response taken from the official AWS documentation. This approach directly tells us the reliability of our retrieval layer and the accuracy of the final generated output.

In addition to evaluation metrics, I also developed **Streamlit interface** that is going to be the final step in this project and is going to be the face of the AWS Architect Assistant. The interface integrates seamlessly with the backend, allowing users to enter queries, receive grounded responses, view retrieved sources, and maintain session histories. I implemented layout elements, styling, page configuration, and real-time interaction logic to ensure that the assistant feels responsive and intuitive. This interface serves as the bridge between complex backend retrieval logic and practical end-user usability, enabling users to interact with the model in a clean, efficient, and professional environment.

In addition to these, I also helped out in designing data ingestion workflow, which prepares the foundation for entire RAG system. I helped in designing and refining the logic for **filtering logic** used to clean the raw AWS PDF dataset by removing low-value chunks such as table-of-contents pages, copyright notices, and short or noisy text segments. This step ensured that only meaningful, high-quality documentation entered the downstream embedding and retrieval pipelines

---

III. RESULTS OF MY WORK

EVALUATION RESULTS:

I constructed a set of five AWS questions spanning different service domains—including Lambda, DynamoDB, EC2, S3, and IAM. These Questions are:

Q1. "What is the maximum execution time for an AWS Lambda function?"

Q2. "What is the difference between a Scan and a Query in DynamoDB?"

Q3. "How do I create an S3 bucket?"

Q4. "What is the difference between On-Demand and Spot Instances?"

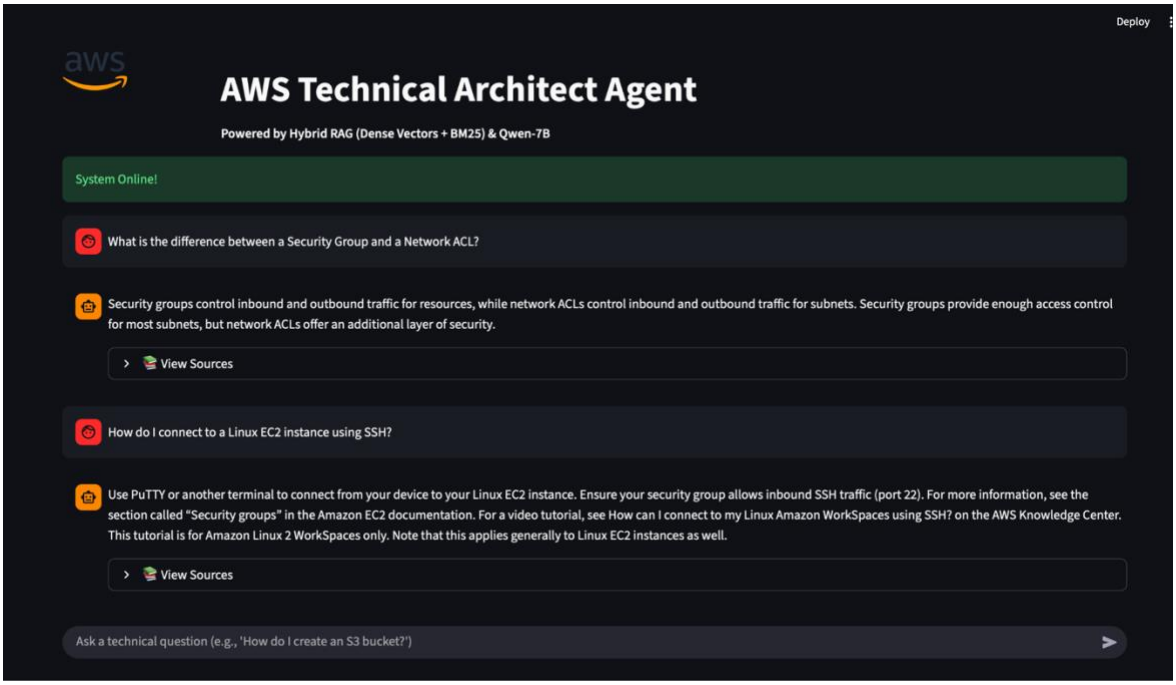
Q5."What is the AWS root user?"

For each question, I manually prepared a answer with reference to AWS documentations. Both the actual answers and the model's responses were embedded using the same **BAAI/bge-m3** embedding model to ensure consistency in vector space representation. The cosine similarity between the ground truth embedding (A) and model-generated embedding (B) was calculated using the equation:

$$\text{Cosine Similarity} = \frac{A \cdot B}{\|A\| \, \|B\|}$$

Here A represents the embedding of the ground truth answer and B represents the embedding of the retrieved context. A value closer to **1** indicates strong alignment in meaning, while values closer to **0** indicate semantic divergence, and a score closer to **-1** indicates a strong opposition in the meaning. Technical documentation often includes long, descriptive text and a perfect score is unrealistic. Therefore we selected **0.60** as our performance target, which is closer to industry practice for RAG systems. But the system surpassed this benchmark in every case, producing scores of 0.7707, 0.6745, 0.7777, 0.7575, and 0.7715, resulting in an average similarity of **0.7504**. These results show that the hybrid search mechanism—combining BM25, vector retrieval, and Reciprocal Rank Fusion successfully captured relevant context across diverse AWS topics, validating the strength and consistency of the retrieval pipeline.

STREAMLIT INTERFACE:



As shown in the image above, the final product comes a Streamlit application which is intuitive, conversational and easy to use interface for users. This application displays streaming responses along with source citations

through expandable panels, which gives an option to users to also see the direct citation from AWS documents. transparency and trust in the model's outputs.

IV. SUMMARY

- **Designed and implemented the evaluation framework** using cosine similarity, I measured semantic alignment between model generated outputs and answers from AWS documentation.
- **Computed and analyzed similarity scores**, achieved an average performance of 0.7504—well above the 0.60 target threshold used in practical RAG systems.
- **Developed the Streamlit interface**, which includes layout, styling, session handling, and real-time interaction logic to deliver a smooth user experience.
- **Assisted in building the data ingestion pipeline** by refining filtering rules that removed noisy, low-value text and ensured a clean, high-quality dataset for indexing.

V. PERCENTAGE OF CODE LINES

Now let's look at the percentage of code I found from Internet:

- Number of lines of code present in the part of project i contributed (excluding 'import' statements): 140
- Number of lines i found from Internet: 110
- Number of lines i modified: 50
- Number of lines i added: 30

The percentage is going to be: ((110 - 50) / (110 + 30)) * 100 = **42.85%**

VI. REFERENCES

1. X. Han Lù (2024). BM25 for Python: Achieving High Performance While Simplifying Dependencies with BM25S.
2. Khattab, O., & Zaharia, M. (2020). ColBERT: Efficient and Effective Passage Search via Contextualized Late Interaction over BERT.
3. Lewis, P., Perez, E., Piktus, A., et al. (2020). Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks.
4. Dao, T., Padhy, S., Chung, H. W., et al. (2024). Textbooks Are All You Need II: RAG and the Next Generation of Retrieval-Augmented Models.
5. Dr. Julija (2024). How I Built a Simple Retrieval-Augmented Generation (RAG) Pipeline.
6. Dataset: aws-public-pdf-chunked-dataset.
7. Github repo of this project: Final-Project-Group-BotBuilders.
