Individual Final Project Report

Project : AWS CloudGuide – Your Cloud Architecture Assistant
Student Name: Jasreen Kaur Mehta

---

1. Introduction

This project involved building AWS CloudGuide, a Retrieval-Augmented Generation (RAG) system designed to improve the accessibility of AWS documentation by generating grounded, citation-backed answers. The team collaborated on dataset ingestion, preprocessing, indexing, retrieval, and user interface development.

My individual contribution focused on building the LLM Answer Generation Pipeline, which converts retrieved AWS documentation into accurate, grounded responses. I also developed much of the backend logic that manages inference, session handling, prompt construction, and integration with the retrieval layer.

This report describes my individual responsibilities, the algorithms behind the LLM component, the backend implementation, and the results generated from the final working system. It concludes with reflections on what I learned and how the system could be improved in the future.

---

2. Background & Algorithmic Foundations

My part of the project centered on implementing the generation layer of the RAG pipeline. In a hybrid retrieval setting, the role of the LLM is not to invent answers, but to synthesize retrieved context into clear and accurate explanations.

2.1 Prompt Construction Strategy

I designed the system prompt to enforce factual grounding:

“You are an AWS documentation assistant.
Use ONLY the provided documentation to answer.
If the answer is not present in the context, say "The provided documentation does not include that information."”

The final prompt includes:

- The grounding instruction
- The retrieved AWS text
- The user’s question

This format ensures transparency and reduces hallucinations.

2.2 LLM Model & Inference

The model used for this project is Qwen2.5-7B-Instruct, loaded in half-precision (FP16) on GPU for speed and memory efficiency.

The backend initializes the model using HuggingFace's pipeline:

- max_new_tokens = 1024
- temperature = 0.1
- top_p = 0.9

Lower temperature ensures deterministic, documentation-grounded responses.

2.3 Query Rewriting Module

I implemented a dedicated LLM-based query rewriting system in the backend. This feature improves retrieval when users ask vague follow-up questions like:
"How do I configure it?"

The module rewrites such questions into complete versions using prior chat history:
"How do I configure Amazon Q?"
This significantly improves retrieval accuracy and prevents ambiguous answers.

---

3. Detailed Description of My Work

One of my primary responsibilities in the project was developing the backend system that orchestrates the complete LLM-driven answer generation workflow. This backend serves as the bridge between the retrieval engine, the language model, and the user interface, ensuring that each question moves through the pipeline in a structured, efficient, and grounded manner.

The backend performs several key functions:

3.1 Model Initialization and Resource Management

I implemented the logic to load the Qwen2.5-7B-Instruct model in FP16 precision with automatic device placement. This optimization ensures that the model uses GPU resources efficiently and minimizes memory overhead, enabling smooth inference even on limited hardware.

This included:

- Loading both the tokenizer and model with appropriate flags
- Ensuring deterministic behavior by setting low temperature defaults
- Applying chat templates to ensure model instructions are formatted correctly
- Managing warm-up calls to reduce first-token latency

3.2 Prompt Assembly and Grounding Enforcement

A major component I built was the prompt construction subsystem, which merges retrieved AWS documentation chunks with the user query inside a structured prompt. This included:

- Dynamically injecting the top RRF-ranked chunks
- Removing duplicate or low-quality chunks
- Truncating overly long segments to stay within context window limits
- Adding grounding instructions to prevent hallucinations

The backend constructs the final prompt in three layers:

1. System Instruction – defines strict grounding rules
2. Retrieved Context – multiple ordered, properly separated chunks
3. User Query – rewritten (if applicable) and cleaned

This ensures the LLM receives clean, optimized input every time.

3.3 Retrieval-to-Generation Orchestration

My backend logic performs the complete orchestration between retrieval and the LLM. This involves:

- Calling BM25, vector search, and RRF fusion
- Extracting the top *k* documents
- Cleaning and formatting them for prompt insertion
- Ensuring empty or irrelevant results trigger fallback mechanisms (e.g., apologizing and stating missing info)

This orchestration layer guarantees that the LLM always operates on high-quality, relevant context.

3.4 Post-Processing and Citation Injection

After the LLM generates a response, the backend performs multiple post-processing steps that I implemented:

- Cleaning repeated tokens and hallucinated prefixes
- Adding structured citations using the metadata from the retrieved chunks
- Ensuring that the final answer text is readable and properly segmented
- Stripping out any model-internal formatting that should not be exposed to users

Citation injection was particularly important because AWS CloudGuide is designed as a transparent AI assistant where users can verify the origin of claims.

3.5. Session Management and Multi-Turn Support

I implemented the session-level logic that allows users to have multi-turn interactions. This includes:

- Maintaining conversation state across requests
- Passing previous user questions and system responses
- Tracking the “topic entity” needed for query rewriting
- Ensuring that every turn uses a fresh, context-aware prompt

This makes the LLM behave like a knowledgeable AWS assistant instead of a stateless text generator.

3.6 Integration with the Streamlit User Interface

Although the frontend interface was handled by another teammate, I built the backend hooks required to allow real-time interaction. This involved:

- Returning model outputs in Streamlit-compatible structures
- Handling Δ-style partial updates if needed
- Managing exceptions and providing user-friendly fallback messages

This integration ensures seamless interaction between our backend logic and the UI where users ask AWS questions.

3.7 Inference Parameter Control and Stability Tuning

A subtle but important part of my backend work was experimenting with inference hyperparameters. Through testing, I identified the ideal settings for grounded AWS documentation responses:

- temperature = 0.1 for stable, non-creative answers
- top_p = 0.9 for controlled sampling
- max_new_tokens = 1024 to allow full explanations without overflow

These settings were selected after iterative debugging and comparison of model outputs during different tuning phases.

---

4. Results

After integrating the LLM pipeline with retrieval and the user interface, the system produced:

4.1 Clear, Structured, Grounded Answers

The model consistently generated:

- accurate explanations of AWS services
- step-by-step instructions for AWS tasks
- comparisons between similar AWS features
- answers supported by citations pointing back to the documentation

4.2 Strong Alignment With Retrieved Documentation

The LLM rarely hallucinated due to:

- strict grounding instructions
- curated context chunks
- low-temperature decoding

4.3 High-Quality User Experience

In the Streamlit app, responses:

- appeared cleanly formatted
- included expandable citations
- streamed in real time
- supported multi-turn conversations

4.4 Technical Advantages Observed

- FP16 inference on Qwen2.5-7B was fast and efficient
- Hybrid retrieval improved input quality to LLM
- Citations increased trust and explainability

Overall, the LLM pipeline worked reliably and produced high-quality answers across a range of AWS topics.

---

5. Summary and Conclusions

Through this project, I developed the full LLM generation layer, including backend inference logic, grounded prompting, query rewriting, and answer formatting. The system successfully transformed AWS documentation into conversational, easy-to-understand, and citation-backed responses.

Key takeaways:

- Retrieval quality heavily impacts LLM answer strength
- Grounded prompting significantly reduces hallucinations
- Query rewriting improves clarity and relevance in multi-turn chats
- Smaller models like Qwen2.5-7B can perform very well when retrieval is strong

Future improvements:

- Experiment with larger LLMs (Qwen 14B, Llama 70B)
- Add code snippet retrieval for AWS CLI/SDK examples
- Introduce follow-up refinement (e.g., “simplify this explanation”)
- Use multi-query retrieval for even stronger context recall

Overall, this project deepened my understanding of retrieval-augmented generation systems and the importance of grounded LLM design in building reliable AI assistants.

---

6. Code Percentage Calculation

The implementation of the LLM pipeline and backend inference layer relied on Hugging Face Transformers for core abstractions such as model loading, tokenization, and generation utilities. However, the logic responsible for grounding responses, assembling prompts, rewriting queries, integrating retrieval outputs, and managing chat sessions was custom-developed.

Code Sources:

- Hugging Face documentation (loading Qwen2.5, pipeline initialization)
- Transformers chat templates and generation API examples
- Standard Python utilities for text formatting and state handling

My Contribution:

- Custom prompt construction and grounding instructions
- Query rewriting mechanism for resolving ambiguous follow-up questions
- Backend logic connecting hybrid retrieval to LLM inference
- Context assembly, duplication filtering, and safe text-preprocessing steps
- Citation injection and response formatting
- Session state management to support multi-turn conversations
- Hyperparameter tuning (temperature, max tokens, top-p) for stable generation

Calculation:

- Total Lines of Code (LLM + backend file): ~319 lines
- Lines adapted from external tutorials/documentation: ~100 lines
  (model initialization, pipeline boilerplate, import patterns)
- Lines written from scratch: ~219 lines
  (prompt logic, rewriting logic, inference orchestration, formatting, session logic)

Percentage of Own Code:

Approximately 69% of the LLM pipeline and backend code was written by me, with the remaining portion adapted from library documentation or standard boilerplate patterns.

---

7. References

- AWS Official Documentation, 2025
- HuggingFace Transformers Documentation
- Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.-t., Rocktäschel, T., Riedel, S., & Kiela, D. (2020). *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks.* In *Advances in Neural*

Information Processing Systems 33 (NeurIPS 2020). https://proceedings.neurips.cc/paper/2020/file/6b493230205f780e1bc26945df7481e5-Paper.pdf