Junior AI Engineer Assignment: Fine-Tuned RAG Chatbot with Streaming Responses – Amlgo Labs

Amlgo Labs is hiring for the position of Junior AI Engineer, and as part of our selection process, you are required to complete the following technical assignment.

The goal of this assessment is to build the project described below, demonstrating your applied knowledge in Natural Language Processing (NLP), LLMs, retrieval systems, and deployment of AI applications. This will test your ability to work end-to-end—from data preparation to model integration and serving the solution via a user-friendly interface.

1. Overview & Objectives

Goal:
Build and demo an AI-powered chatbot capable of answering user queries based on a provided document set (e.g., Terms & Conditions, Privacy Policies, Legal Contracts).

The solution must implement a Retrieval-Augmented Generation (RAG) pipeline using a fine-tuned or instruction-optimized open-source LLM and a vector database.

The chatbot must support real-time streaming responses via a Streamlit interface.

Key Learning Outcomes:

- Document preprocessing, chunking, and semantic embedding
- Building and querying a vector database (FAISS, Chroma, or Qdrant)
- Fine-tuning or instructing an open-source LLM (e.g., Mistral, LLaMA, Zephyr)
- Implementing the complete RAG pipeline (retriever + generator)
- Building a Streamlit chatbot interface with streaming responses

2. Assignment Tasks

2.1 Document Preparation & Ingestion

Document Set:
You will be provided with a document (~10,500+ words). Your task is to process the document and prepare it for retrieval.

Steps:

- Clean and format the text if needed (remove headers, footers, HTML, etc.)
- Chunk the documents into 100–300 word segments using sentence-aware splitting
- Generate embeddings using a pre-trained model like all-MiniLM, bge-small-en, or similar
- Store the embeddings in a vector database (e.g., FAISS, Chroma, Qdrant)

2.2 Model Fine-Tuning / Prompt Tuning

Model Selection:
Use a small, open-source LLM such as mistral-7b-instruct, llama-3, or zephyr-7b.

Tuning Approach:
Design a high-quality prompt template for injecting retrieved chunks + user queries.

2.3 RAG Pipeline Implementation

Retriever:

- Perform semantic search on the indexed chunked documents using the vector DB

Generator:

- Inject retrieved chunks + user query into the prompt
- Ensure the final output is factual and grounded in the retrieved documents

Response Handling:

- Output should be streamed in real-time (token-by-token or sentence-by-sentence)
- Include relevant source chunks or references used to generate the answer

2.4 Streamlit Chatbot Deployment with Streaming

Features Required:

- User input field for natural language queries
- Real-time streaming model response
- Display of source text passages used to generate the answer
- Sidebar or footer showing:
    - Current model in use
    - Number of chunks or indexed documents
- Clear chat/reset functionality

Note on Deployment:
Submit a public GitHub repository containing:

- All project code
- A clear and detailed README
- Screenshots or screen recording (GIF/video) demonstrating the chatbot running locally

3. Deliverables

Folder Structure:

- /data – provided document files
- /chunks – processed and embedded text segments
- /vectordb – saved vector database
- /notebooks – for preprocessing, tuning, and evaluation
- /src – retriever, generator, and pipeline scripts
- app.py – Streamlit app with streaming support

- requirements.txt

- README.md

README.md Must Contain:

- Clear explanation of the project architecture and flow
- Steps to run preprocessing, create embeddings, and build the RAG pipeline
- Model and embedding choices explained
- Instructions to run the chatbot with streaming response enabled
- Sample queries and output screenshots or a link to a demo video

PDF Report (2–3 Pages):

- Description of document structure and chunking logic
- Explanation of embedding model and vector DB used
- Prompt format and generation logic
- At least 3–5 example queries with responses (highlight success and failure cases)
- Notes on hallucinations, model limitations, or slow responses

Demo:

- Include in README:
    - GIF or video link showing chatbot streaming responses
    - GitHub repository URL with complete source code

4. Evaluation Criteria

Criteria	Weight
Functionality & Integration	30%
Streaming Output Implementation	20%
Code Quality & Modularity	20%
Grounded & Accurate Answers	20%
App Usability & UX	10%

Note on Originality & Plagiarism

All code, prompts, documentation, and analysis must be your own work. You are encouraged to consult public references and official documentation, but copy-pasting from ChatGPT, GitHub, blogs, or other sources without understanding or attribution is strictly prohibited.

We will conduct manual code reviews and may ask you to walk through your implementation and design choices. Any detected plagiarism will result in immediate disqualification.