Bluejay Take-home Interview: Building a RAG-Enabled Voice Agent

Overview

Build a RAG-enabled voice agent that solves a personal problem in your life. Use **LiveKit Cloud** for hosting the server.

Please see this video explanation here as well.

Objectives

- Create a voice agent with LiveKit that you can converse with on a custom (simple) frontend.
- Transcribe speech in real time, and maintain a live transcript.
- Give your agent a personality, **spin a story**. Be creative. **Put time into this part!** I will consider the creativity / story-telling to be your behavioral!
- In the call, **make a tool call of your choice** that fits in with the narrative of your voice agent.
    - EX: If you are building a basketball coach voice agent, tool call could be a list of nearby courts.
- Use RAG to enable your voice agent to answer a question from a large PDF document related to your story
    - EX: If you are building a basketball coach voice agent, use a PDF of a book on Steve Kerr to have your voice agent give advice on optimal OT strategies.
- Bonus points if you deploy your Agent on **AWS** instead of running it locally.

Technical Requirements

1. [Backend] LiveKit Setup

- Deploy a LiveKit server or use LiveKit Cloud for server hosting.
- Build a LiveKit Agent that is capable of joining the room and having a live conversation with the user about a topic of your choice.
- The LiveKit agent can either be hosted locally or on AWS (bonus points for AWS).
- Use a voice agent pipeline with configurable STT, LLM, TTS, VAD.

2. [Backend] RAG Over Uploaded PDF

- Enable your voice agent to retrieve information from a large PDF of your choosing. Ensure the PDF is relevant to your overall voice agent story.
- Once uploaded, your agent should be able to **run Retrieval-Augmented Generation (RAG)** over the content of the PDF and answer questions about it **in real-time voice conversation**.

- I will ask about a **specific fact in a specific chapter**, so a proper RAG setup is essential — not just keyword search or summarization.
- You are **encouraged to use an existing RAG framework** (like LangChain, LlamaIndex, or OpenAI's retrieval tools) instead of building a vector store from scratch.
- This documentation will help you integrate the RAG system into your LiveKit Agent: LiveKit Agents + External Data (Docs)

3. [Frontend] Chat Interface with Real-Time Transcription

- Build a **React-based client** with:
    - "Start Call" button.
    - **Live transcript** display that updates as participants speak.
    - "End Call" button.
    - Optionally, add a **PDF upload button** to the frontend *if* it fits your UX flow. No points lost if this feature is not there.

Constraints

- Must use **LiveKit**.
- Use any AI tools you need/want, just document them.

Deliverables

Submit a single **Git repository** containing:

1. **Full Application Code**
    - LiveKit Python backend (including RAG logic).
    - React frontend.
    - Vector store.
2. **README.md**
    - A short **design document** explaining:
        - How your system works end-to-end.
        - How RAG was integrated.
        - The tools/frameworks used.
    - **Setup instructions** to run the project locally or on the web.
3. **Design Decisions / Assumptions**
    - Describe:
        - Any trade-offs or limitations.
        - Hosting assumptions.
        - RAG assumptions (e.g., vector DB choice, chunking strategy, frameworks).
        - LiveKit agent design.

Submission

Please **share the GitHub repo with `farazs27@gmail.com`** when you're done.

Please also send an email with a video demo of your submission to rohan@getbluejay.ai and faraz@getbluejay.ai.