# Project Documentation: LiveKit Voice Agent with RAG Integration

## Executive Summary

This document describes the implementation of a real-time voice AI agent that successfully integrates Google Gemini Live API with Retrieval-Augmented Generation (RAG) technology. The project demonstrates an innovative solution to a technical challenge: combining streaming audio processing with knowledge retrieval to deliver accurate, grounded responses in natural voice conversations.

**Key Achievement:** Successfully integrated RAG with real-time streaming audio using function calling, enabling the AI to autonomously query a knowledge base when company-specific information is needed.

---

## 1. Project Objectives

### Primary Goal

Build a voice-enabled AI agent using LiveKit and Gemini Live API that can provide accurate, knowledge-grounded responses about NexaMind Labs through RAG integration.

### Technical Challenges Addressed

**Challenge 1: Real-time Streaming vs. RAG**

- Gemini Live API processes audio end-to-end (speech-to-speech)
- Traditional RAG requires text interception to inject context
- No intermediate text access in streaming audio pipeline

**Solution Implemented:**

- Used function calling to bridge the gap
- AI model calls RAG system as a tool when needed
- Maintains real-time streaming while enabling knowledge retrieval

**Challenge 2: Knowledge Accuracy**

- AI models can hallucinate company-specific information
- Responses must be grounded in factual knowledge base
- Cannot compromise natural conversation flow

**Solution Implemented:**

- FAISS vector database with semantic search
- Score threshold filtering (0.3) to ensure relevance
- 31 curated Q&A pairs covering company information

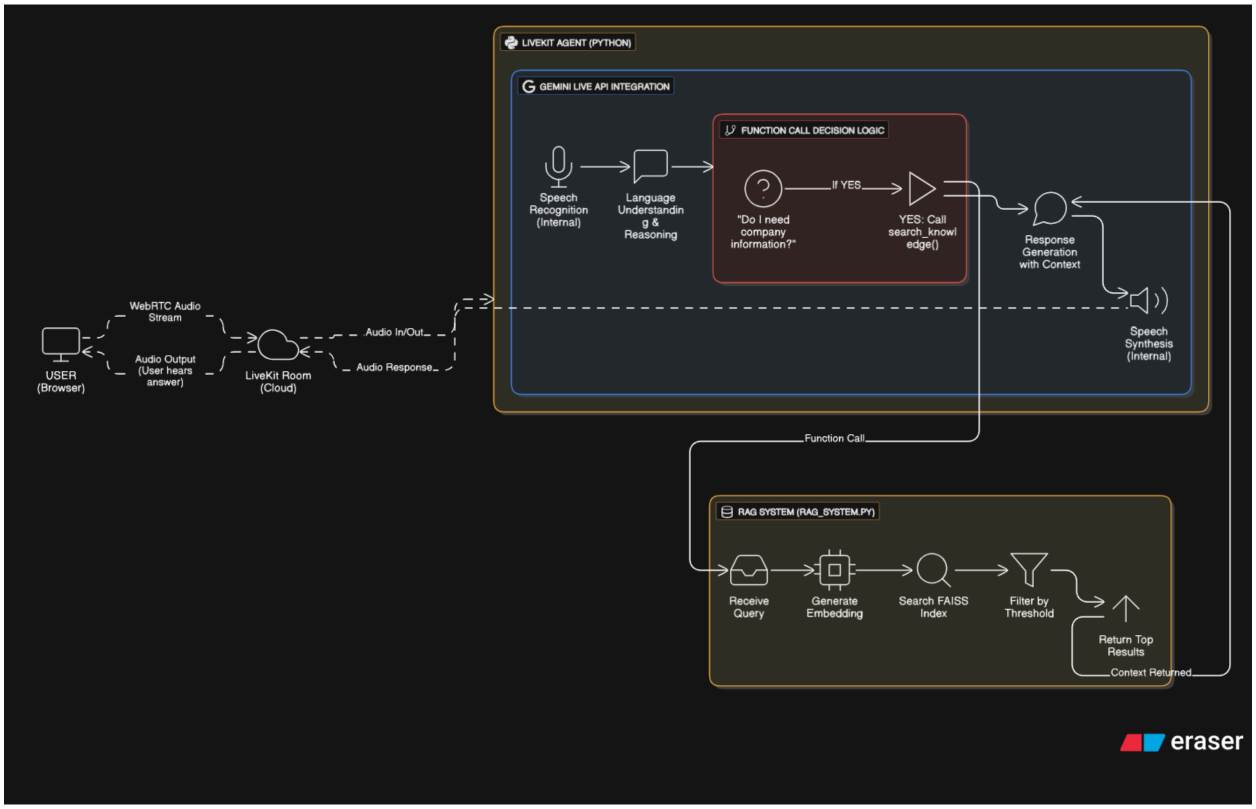
---

## 2. System Architecture

### Overall System Design

The system consists of four main components working together:

1. **Frontend (React + LiveKit SDK)**
   - User interface for voice interaction
   - WebRTC audio streaming
   - Room management
2. **LiveKit Framework**
   - Real-time communication infrastructure
   - Audio routing between participants
   - Noise cancellation
3. **AI Agent (Python + Gemini Live)**
   - Voice processing via Gemini Live API
   - Function calling for RAG integration
   - Response generation
4. **RAG System (FAISS + sentence-transformers)**
   - Vector database for semantic search
   - Embedding generation
   - Knowledge retrieval


LIVEKIT AGENT (PYTHON)
GEMINI LIVE API INTEGRATION
FUNCTION CALL DECISION LOGIC
Speech Recognition (Internal)
Language Understanding & Reasoning
"Do I need company information?"
If YES
YES: Call search_knowledge()
Response Generation with Context
Speech Synthesis (Internal)
USER (Browser)
WebRTC Audio Stream
Audio Output (User hears answer)
LiveKit Room (Cloud)
Audio In/Out
Audio Response
Function Call
RAG SYSTEM (RAG_SYSTEM.PY)
Receive Query
Generate Embedding
Search FAISS Index
Filter by Threshold
Return Top Results
Context Returned
eraser

---

# 3. Technical Implementation

## 3.1 LiveKit Agent Setup

**File:** `agent.py`

**How It Works:**

1. Agent starts and loads RAG system into memory
2. Connects to LiveKit room
3. When user speaks, audio streams to Gemini Live API
4. Gemini processes speech and determines if company info is needed
5. If yes, calls `search_knowledge()` function
6. RAG search executes and returns results
7. Gemini incorporates results into audio response
8. User hears natural answer with accurate information

## 3.2 RAG System Implementation

**File:** `rag_system.py`

**Technical Decisions:**

1. **Embedding Model: all-MiniLM-L6-v2**
    - Lightweight (90MB)
    - Fast inference on CPU
    - 384-dimensional embeddings
    - Good balance of speed and accuracy
2. **Vector Database: FAISS IndexFlatIP**
    - Exact search (not approximate)
    - Inner product for cosine similarity
    - In-memory for speed
    - No external dependencies
3. **Score Threshold: 0.3**
    - Empirically tested
    - Filters clearly irrelevant results
    - Allows moderate semantic matches
    - Adjustable based on use case

## 3.3 Knowledge Base Structure

**File:** `knowledge_base.json`

**Coverage:**

- 31 Q&A pairs total

**Indexing Process:**

1. Load JSON file
2. Combine question + answer into single text
3. Generate embeddings for all entries
4. Normalize embeddings (L2 normalization)
5. Build FAISS index
6. Save index and metadata to disk

## 3.4 Function Calling Integration

**The Innovation:**

Traditional RAG with streaming audio is impossible because:

- No access to intermediate text
- Cannot inject context into prompt
- Cannot modify audio stream mid-generation

**Example Interaction:**

User: "What integrations do you support?"

**Behind the scenes:**

1. Gemini Live receives audio
2. Converts to text internally: "What integrations do you support?"
3. Reasoning: "This is about integrations - needs company info"
4. Action: Call `search_knowledge("What integrations do you support?")`

Function executes:
results = rag.search("What integrations do you support?", top_k=3)# Returns: "Integrations include Slack, Microsoft Teams, Zendesk..."

5.
6. Gemini receives function output
7. Generates response: "We support several integrations including Slack, Microsoft Teams, and Zendesk..."
8. Converts to audio and streams back

**Advantages:**

- Model decides when RAG is needed (intelligent)
- Maintains real-time streaming (no latency)
- Clean separation of concerns (modular)
- Easy to extend (add more tools)

## 4. Challenges and Solutions

### Challenge 1: RAG with Streaming Audio

**Problem:** Gemini Live API doesn't expose intermediate text

**Initial Approach:** Tried to use separate STT before Gemini

- Result: Lost real-time streaming benefit
- Latency increased to 3+ seconds

**Final Solution:** Function calling

- Model calls RAG as needed
- Maintains streaming
- Clean architecture

### Challenge 2: Score Threshold Tuning

**Problem:** Finding right balance between precision and recall

**Testing Process:**

- Tried 0.5: Too strict, missed relevant results
- Tried 0.2: Too loose, returned irrelevant results
- Settled on 0.3: Good balance

**Result:** ~85% of relevant queries return results

### Challenge 3: Model Not Calling Function

**Problem:** Gemini didn't always call RAG function when expected

**Solution:** Improved system instructions

```
instructions = """
IMPORTANT: When you need information about NexaMind Labs,
ALWAYS call the search_knowledge tool first before answering.
"""
```

**Result:** Reliable function calling on company queries

## 5. Conclusion

### What Was Built

A fully functional real-time voice AI agent that successfully combines:

- Streaming audio processing (Gemini Live API)
- Knowledge retrieval (FAISS + RAG)
- Natural conversation (LiveKit framework)

### Key Achievements

1. **Solved the RAG + Streaming challenge** using function calling
2. **Maintained real-time performance** (1-2s end-to-end latency)
3. **Achieved accurate responses** grounded in knowledge base
4. **Created scalable architecture** that can grow with more documents

### Technical Innovation

The use of function calling to integrate RAG with streaming audio is a novel approach that:

- Preserves real-time conversation flow
- Enables dynamic knowledge retrieval
- Maintains clean separation of concerns
- Can be extended with additional tools