# **Voice Assistant with LiveKit and OpenAI**

## **1. Overview**

This project is a **real-time voice assistant** built using **LiveKit, OpenAI, Deepgram, and Flask**. It performs speech-to-text (STT), natural language processing (NLP) with OpenAI’s LLM, and text-to-speech (TTS) for responses. The assistant ensures responses stay within a 60-second limit by trimming excessive text from the center.

## **2. Setup Process**

### **2.1 Prerequisites**

Before setting up, ensure you have the following installed:

* **Python 3.8+**
* **Virtual environment** (recommended)
* **Dependencies**:
  + requests
  + asyncio
  + flask
  + flask\_cors
  + livekit
  + deepgram
  + openai
  + Silero
  + Cartesia
  + groq
* **Ngrok** (for exposing local APIs)

### **2.2 Installation Steps**

#### **Step 1: Clone the Repository**

bash

git clone <repository\_url>

cd <project\_directory>

#### 

#### 

#### **Step 2: Create a Virtual Environment**

bash

python -m venv venv

source venv/bin/activate # Windows: `venv\Scripts\activate`

#### **Step 3: Install Dependencies**

bash

pip install -r requirements.txt

#### **Step 4: Set Up Environment Variables**

Create a .env file and configure API keys:

env

LIVEKIT\_URL='wss://text-to-speech-3x2as51y.livekit.cloud'

LIVEKIT\_API\_KEY='APISVpgrBE2FxSm'

LIVEKIT\_API\_SECRET='f490BT8lkKMf0LAQb5dzbJ9lVDSsHjT4iEOHeaSxyn2A'

DEEPGRAM\_API\_KEY='e87b679517d41225f78f883aaa41ebb65c344d41’

GROQ\_API\_KEY='gsk\_K5bGp5vluQ48Fgs7MwjjWGdyb3FYtNE5h6MMYnxKkchZOSHVdLYwCARTESIA\_API\_KEY='sk\_car\_Y8dTQZs3cAZA5BU6xmg7k'

#### **Step 5: Start the Flask API**

bash

python api.py

#### **Step 6: Expose API using Ngrok**

bash

ngrok http 5000

## **3. Code Implementations**

### **3.1 Flask API for Text Validation**

The API ensures that the assistant's responses fit within a 60-second limit by trimming the center part if necessary.

**Python code**

from flask import Flask, request, jsonify

from flask\_cors import CORS

app = Flask(\_\_name\_\_)

CORS(app)

def estimate\_audio\_length(text, words\_per\_second=2):

words = text.split()

return len(words) / words\_per\_second

def trim\_text(text, max\_duration=60, words\_per\_second=2):

words = text.split()

max\_words = max\_duration \* words\_per\_second

if len(words) <= max\_words:

return text

start = (len(words) - max\_words) // 2

end = start + max\_words

return ' '.join(words[start:end])

@app.route('/validate-audio-length', methods=['POST'])

def validate\_audio():

data = request.json

text = data['text']

audio\_length = data.get('audio\_length', estimate\_audio\_length(text))

if audio\_length > 60:

text = trim\_text(text)

return jsonify({'validated\_text': text, 'audio\_length': audio\_length})

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

### 

### **3.2 LiveKit Voice Assistant**

The assistant:

* Listens for audio input.
* Converts speech to text using **Deepgram**.
* Generates responses using **OpenAI LLM**.
* Ensures responses fit within 60 seconds.
* Converts responses back to speech using **OpenAI TTS**.

**Code:**

import asyncio

import logging

import requests

from dotenv import load\_dotenv

from livekit import rtc

from livekit.agents import (

AutoSubscribe,

JobContext,

JobProcess,

WorkerOptions,

cli,

llm as lm,

metrics,

)

from livekit.agents.pipeline import VoicePipelineAgent

from livekit.plugins import cartesia, deepgram, openai, silero

from livekit.plugins.openai import llm

# Load environment variables

load\_dotenv()

# Configure logger

logger = logging.getLogger("voice-assistant")

# Initialize LLM with Groq model

groq\_llm = llm.LLM.with\_groq(

model="llama3-8b-8192",

temperature=0.1,

)

def prewarm(proc: JobProcess):

"""Preload VAD model."""

proc.userdata["vad"] = silero.VAD.load()

async def entrypoint(ctx: JobContext):

"""Main entrypoint for the voice assistant."""

initial\_ctx = lm.ChatContext().append(

role="system",

text=(

"You are a voice assistant created by LiveKit. Your interface with users will be voice. "

"You should use short and concise responses, avoiding unpronounceable punctuation."

),

)

def validate\_text(text: str):

"""Trim content if it exceeds 60 seconds using an external API."""

response = requests.post(

'https://9945-183-82-34-206.ngrok-free.app/validate-audio-length',

json={'text': text}

)

return response.json().get('validated\_text', text) if response.status\_code == 200 else text

logger.info(f"Connecting to room {ctx.room.name}")

await ctx.connect(auto\_subscribe=AutoSubscribe.AUDIO\_ONLY)

participant = await ctx.wait\_for\_participant()

logger.info(f"Starting voice assistant for participant {participant.identity}")

dg\_model = "nova-3-general" if participant.kind != rtc.ParticipantKind.PARTICIPANT\_KIND\_SIP else "nova-2-phonecall"

agent = VoicePipelineAgent(

vad=ctx.proc.userdata["vad"],

stt=deepgram.STT(model=dg\_model),

llm=groq\_llm,

tts=cartesia.TTS(),

chat\_ctx=initial\_ctx,

before\_tts\_cb=lambda assistant, text: text, # Call only once

)

agent.start(ctx.room, participant)

usage\_collector = metrics.UsageCollector()

@agent.on("metrics\_collected")

def \_on\_metrics\_collected(mtrcs: metrics.AgentMetrics):

metrics.log\_metrics(mtrcs)

usage\_collector.collect(mtrcs)

async def log\_usage():

summary = usage\_collector.get\_summary()

logger.info(f"Usage: ${summary}")

ctx.add\_shutdown\_callback(log\_usage)

chat = rtc.ChatManager(ctx.room)

async def answer\_from\_text(txt: str):

"""Generate response using LLM and validate text."""

chat\_ctx = agent.chat\_ctx.copy()

chat\_ctx.append(role="user", text=txt)

response\_text = ""

async for chunk in agent.llm.chat(chat\_ctx=chat\_ctx):

if chunk.choices and chunk.choices[0].delta.content:

response\_text += chunk.choices[0].delta.content

validated\_text = validate\_text(response\_text) # Call API only once

await agent.say(validated\_text)

@chat.on("message\_received")

def on\_chat\_received(msg: rtc.ChatMessage):

if msg.message:

asyncio.create\_task(answer\_from\_text(msg.message))

await agent.say("Hey, how can I help you today?", allow\_interruptions=True)

if \_\_name\_\_ == "\_\_main\_\_":

cli.run\_app(

WorkerOptions(

entrypoint\_fnc=entrypoint,

prewarm\_fnc=prewarm,

),

)

## **4. Debugging & Challenges Encountered**

### **4.1 Duplicate API Calls**

* **Issue:** The validation API was being called twice, once in before\_tts\_cb and again in answer\_from\_text().
* **Fix:** Removed API call from before\_tts\_cb to ensure it is only called before speech synthesis.

### **4.2 Long Responses**

* **Issue:** LLM sometimes generated responses that exceeded the 60-second limit.
* **Fix:** Implemented **center-based trimming** to retain context while shortening responses.