App.py

```
import os
import warnings
import requests
import json
from langchain_community.embeddings import HuggingFaceEmbeddings
from langchain_community.vectorstores import FAISS
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain.schema import Document

# Suppress warnings from langchain and other libraries
warnings.filterwarnings("ignore", category=DeprecationWarning)

# Initialize embeddings
embeddings = HuggingFaceEmbeddings(model_name="sentence-transformers/all-
MiniLM-L6-v2")

# Path for storing the FAISS index
FAISS_INDEX_PATH = "vectorstore.faiss"
FAISS_CONFIG_PATH = "vectorstore.pkl"

# Load or initialize the FAISS vector store
def load_vectorstore():
    if os.path.exists(FAISS_INDEX_PATH) and os.path.exists(FAISS_CONFIG_PATH):
        return FAISS.load_local(FAISS_INDEX_PATH, embeddings)
    else:
        # Create a new vector store if no saved store exists
        dummy_text = "This is a dummy text to initialize the vector store."
        return FAISS.from_texts([dummy_text], embeddings)

# Save the vector store to disk
def save_vectorstore(vectorstore):
    vectorstore.save_local(FAISS_INDEX_PATH)

# Initialize the vectorstore
vectorstore = load_vectorstore()

# Function to interact with the LLM server
def get_response(prompt):
    url = "http://localhost:11434/api/generate"
    headers = {"Content-Type": "application/json"}
    data = {"model": "tinydolphin", "prompt": prompt}

    response = requests.post(url, headers=headers, json=data)

    if response.status_code != 200:
```

```
        print("Error:", response.text)
        return "Failed to generate response."

    try:
        lines = response.text.splitlines()
        results = [json.loads(line) for line in lines if line.strip()]
    except json.JSONDecodeError as e:
        print(f"JSON Decode Error: {e}, Response: {response.text}")
        return "Failed to parse server response."

    return ''.join(result["response"] for result in results if "response" in
result)

# Determine if the input is informative or a question
def is_informative(text):
    return not text.strip().endswith('?')

# Process user input
def process_input(text):
    global vectorstore  # Access global vectorstore
    if text.strip().lower() == "dolphin forget everything":
        # Clear the vector store
        vectorstore = FAISS.from_texts(["This is a dummy text to initialize
the vector store."], embeddings)
        save_vectorstore(vectorstore)
        return "Memory cleared. I have forgotten everything."
    elif is_informative(text):
        # Store data in memory
        text_splitter = RecursiveCharacterTextSplitter(chunk_size=1000,
chunk_overlap=200)
        docs = text_splitter.split_text(text)
        docs = [Document(page_content=doc) for doc in docs]
        vectorstore.add_texts(
            [doc.page_content for doc in docs],
            embeddings=[embeddings.embed_query(doc.page_content) for doc in
docs]
        )
        save_vectorstore(vectorstore)  # Save updated vector store
        return "Got it! Storing it in my memory."
    else:
        # Retrieve relevant data and generate response
        docs = vectorstore.similarity_search(text, k=3)
        context = "\n".join(doc.page_content for doc in docs)

        prompt = f"""
        You are an AI language model. Provide a response strictly based on the
given context.
```

```
        Context:
        {context}

        Question:
        {text}

        Answer:
        """
        # print(prompt)
        return get_response(prompt)

# Main loop
if __name__ == "__main__":
    print("Assistant: Hello! How can I help you? (Type 'exit' to quit)")
    while True:
        user_input = input("You: ")
        if user_input.lower() == "exit":
            print("Assistant: Goodbye!")
            break
        response = process_input(user_input)
        print(f"Assistant: {response}")
```

stt.py

```
import os
import time
import speech_recognition as sr
from datetime import datetime

# Initialize recognizer
recognizer = sr.Recognizer()

# Directory and file setup
AUDIO_DIR = "input_audio"
TEXT_FILE = "input.txt"

# Ensure the audio directory exists
os.makedirs(AUDIO_DIR, exist_ok=True)

def save_audio(audio, file_name):
    """Saves the audio to the input_audio folder."""
    file_path = os.path.join(AUDIO_DIR, file_name)
```

```
    with open(file_path, "wb") as f:
        f.write(audio.get_wav_data())
    print(f"Audio saved to {file_path}")
    return file_path

def append_text_to_file(text, file_name):
    """Appends the recognized text to the text file."""
    with open(file_name, "a") as f:
        f.write(text + "\n")
    print(f"Text appended to {file_name}")

def listen_and_process(duration=10):
    """
    Listens for speech and processes audio into text.
    Ends the process after the specified duration.
    """
    # Start timing
    start_time = time.time()

    # Use the default microphone as the audio source
    with sr.Microphone() as source:
        print("Adjusting for ambient noise... please wait")
        recognizer.adjust_for_ambient_noise(source)
        print("Listening for speech...")

        try:
            # Listen for speech with a timeout equal to the remaining duration
            while time.time() - start_time < duration:
                print("Recording audio...")
                remaining_time = duration - (time.time() - start_time)
                if remaining_time <= 0:
                    break

                # Listen for speech
                audio = recognizer.listen(source, timeout=remaining_time,
phrase_time_limit=remaining_time)
                print("Processing audio...")

                # Recognize speech using Google Web Speech API
                text = recognizer.recognize_google(audio)
                print("You said:", text)

                # Generate a unique audio file name with timestamp
                timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
                audio_file_name = f"audio_{timestamp}.wav"

                # Save audio and append text
                save_audio(audio, audio_file_name)
```

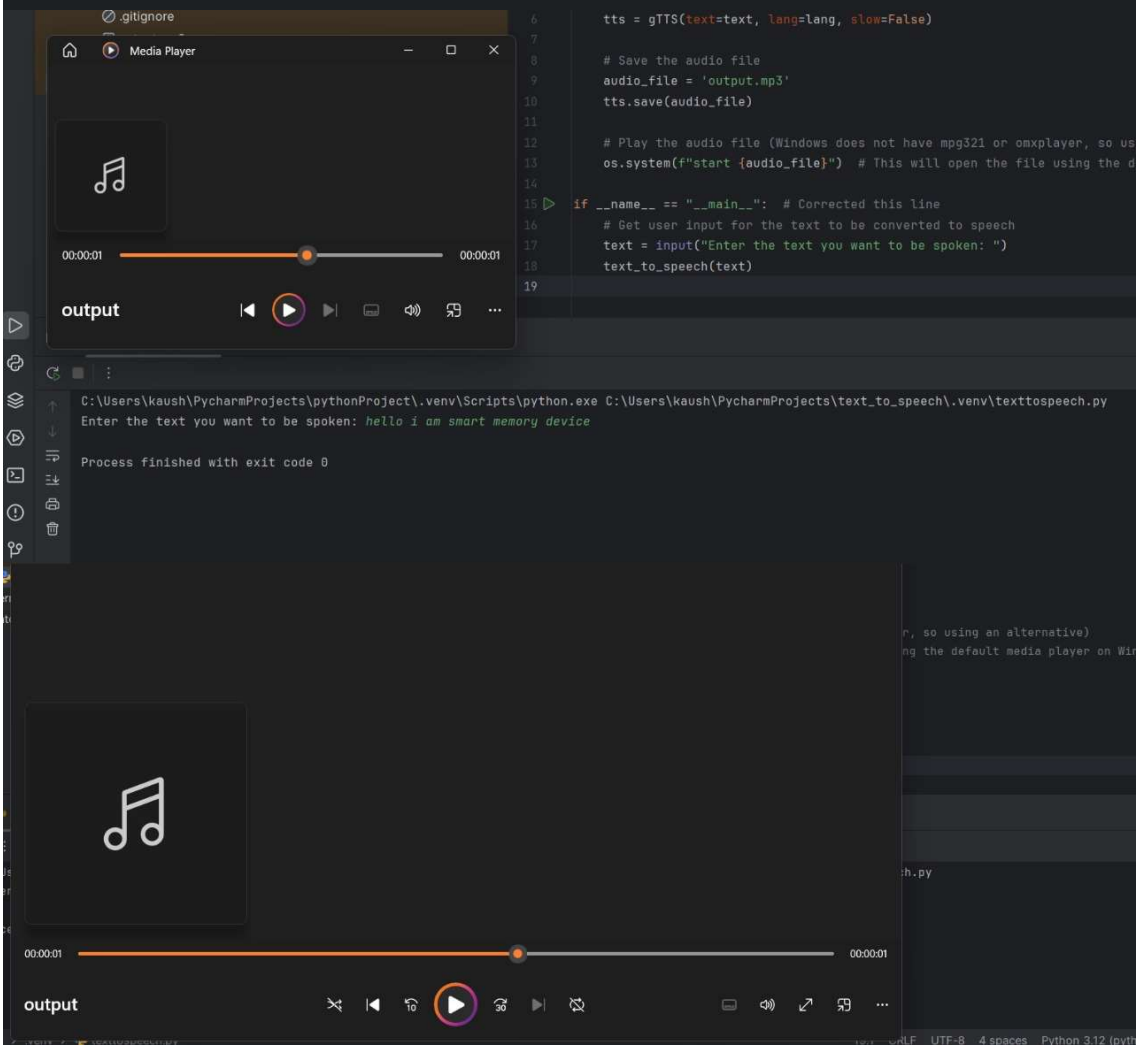
```
                append_text_to_file(text, TEXT_FILE)

        except sr.WaitTimeoutError:
            print("No speech detected within the given time. Exiting...")
        except sr.UnknownValueError:
            print("Could not understand the audio.")
        except Exception as e:
            print(f"An error occurred: {e}")

if __name__ == "__main__":
    print("Starting speech-to-text process...")
    listen_and_process(duration=10)
    print("Listening process ended.")
```

tts.py

```
import pyttsx3
import os
from datetime import datetime

# Directory and file setup
AUDIO_DIR = "output_audio"
TEXT_FILE = "output.txt"

# Ensure the audio directory exists
os.makedirs(AUDIO_DIR, exist_ok=True)

def text_to_speech(text, output_file):
    """
    Converts input text to speech and saves it as a .wav file.

    :param text: The text to convert to speech.
    :param output_file: The path of the output .wav file.
    """
    # Initialize the pyttsx3 TTS engine
    engine = pyttsx3.init()

    # Set properties for the speech
    engine.setProperty('rate', 150)  # Speed of speech
    engine.setProperty('volume', 1.0)  # Volume (0.0 to 1.0)

    # Create the .wav file
    engine.save_to_file(text, output_file)
    engine.runAndWait()
```

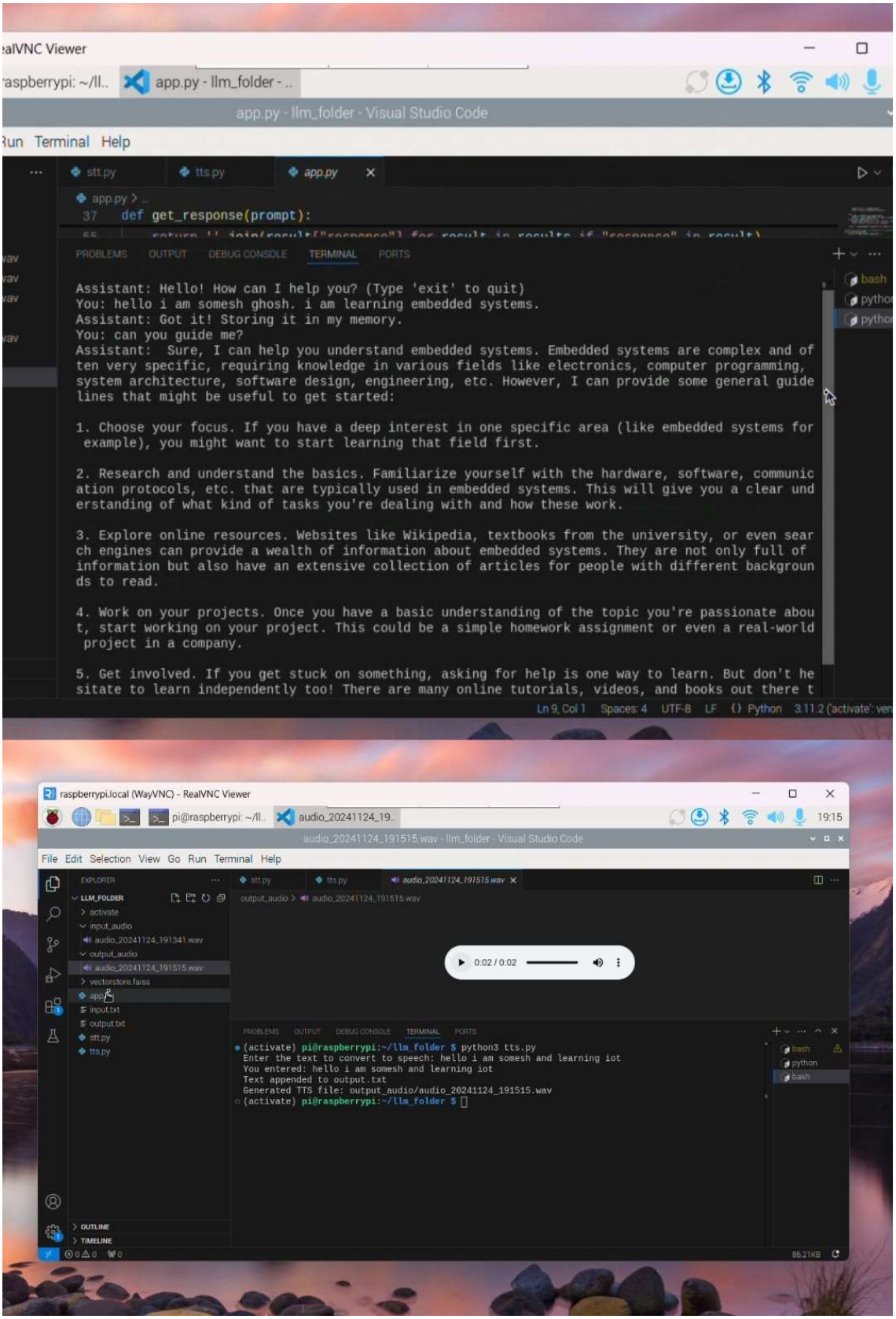
```
    print(f"Generated TTS file: {output_file}")

def append_text_to_file(text, file_name):
    """
    Appends the provided text to the specified text file.

    :param text: The text to append.
    :param file_name: The file to which the text will be appended.
    """
    with open(file_name, "a") as f:
        f.write(text + "\n")
    print(f"Text appended to {file_name}")

if __name__ == "__main__":
    # Take input from the terminal
    text = input("Enter the text to convert to speech: ").strip()

    if not text:
        print("No text provided. Exiting.")
    else:
        # Print the text in the terminal
        print(f"You entered: {text}")

        # Append the text to output.txt
        append_text_to_file(text, TEXT_FILE)

        # Generate unique audio file name with timestamp
        timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
        audio_file_name = f"audio_{timestamp}.wav"
        output_path = os.path.join(AUDIO_DIR, audio_file_name)

        # Convert text to speech and save it
        text_to_speech(text, output_path)
```

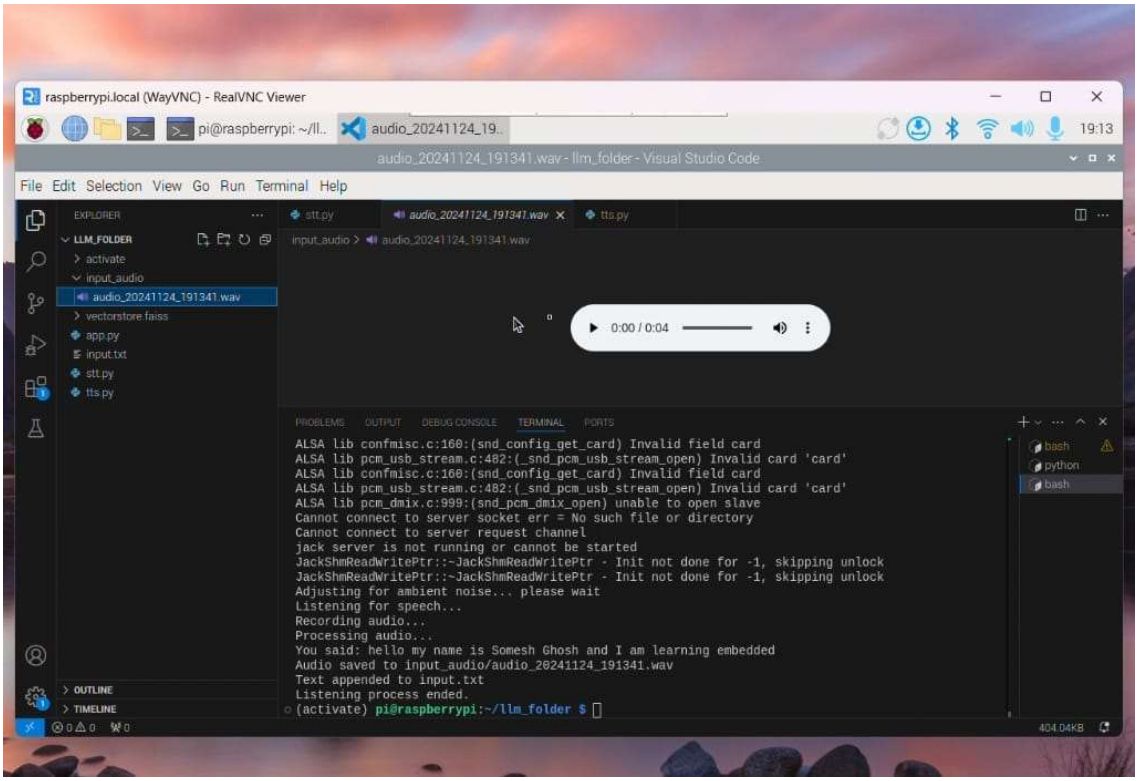
```
Please say your name.
Listening...
Verification score: tensor([0.9425])
User verified successfully.
Listening for commands... (Speak within 5 seconds or say 'stop listening')
You said: hello hello we are in room
Listening for commands... (Speak within 5 seconds or say 'stop listening')
Sorry, I could not understand the audio.
No speech detected for 5 seconds. Stopping.

Process finished with exit code 0

Recording authorized voice... Please speak.
Authorized voice recorded and saved as authorized_voice.wav

Process finished with exit code 0
```



```
Please say your name.
Listening...
Verification score: tensor([0.9425])
User verified successfully.
Listening for commands... (Speak within 5 seconds or say 'stop listening')
You said: hello hello we are in room
Listening for commands... (Speak within 5 seconds or say 'stop listening')
Sorry, I could not understand the audio.
No speech detected for 5 seconds. Stopping.

Process finished with exit code 0
```

```python
import speech_recognition as sr
from speechbrain.pretrained import SpeakerRecognition
import time
import os

# Load the pre-trained speaker recognition model once, with error handling
try:
    speaker_rec_model = SpeakerRecognition.from_hparams(source="speechbrain/spkrec-xvect-voxceleb")
except Exception as e:
    print(f"Error loading speaker recognition model: {e}")
    speaker_rec_model = None


def verify_user():  1 usage
    if not speaker_rec_model:
        print("Speaker recognition model not loaded. Exiting verification.")
        return

    recognizer = sr.Recognizer()
    authorized_file = "authorized_voice.wav"  # The pre-recorded authorized voice file

    if not os.path.exists(authorized_file):
        print("Authorized voice file not found. Please provide a valid file.")
        return

    with sr.Microphone(sample_rate=16000) as source:
        print("Please say your name.")
        recognizer.adjust_for_ambient_noise(source, duration=1)

        start_time = time.time()  # Start a timer for 10 seconds

        while time.time() - start_time < 10:  # Allow a maximum of 10 seconds for the user to speak
```


app.py - llm_folder - Visual Studio Code
stt.py
tts.py
app.py
def get_response(prompt):
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Assistant: Hello! How can I help you? (Type 'exit' to quit)
You: hello i am somesh ghosh. i am learning embedded systems.
Assistant: Got it! Storing it in my memory.
You: can you guide me?
Assistant: Sure, I can help you understand embedded systems. Embedded systems are complex and often very specific, requiring knowledge in various fields like electronics, computer programming, system architecture, software design, engineering, etc. However, I can provide some general guidelines that might be useful to get started:
1. Choose your focus. If you have a deep interest in one specific area (like embedded systems for example), you might want to start learning that field first.
2. Research and understand the basics. Familiarize yourself with the hardware, software, communication protocols, etc. that are typically used in embedded systems. This will give you a clear understanding of what kind of tasks you're dealing with and how these work.
3. Explore online resources. Websites like Wikipedia, textbooks from the university, or even search engines can provide a wealth of information about embedded systems. They are not only full of information but also have an extensive collection of articles for people with different backgrounds to read.
4. Work on your projects. Once you have a basic understanding of the topic you're passionate about, start working on your project. This could be a simple homework assignment or even a real-world project in a company.
5. Get involved. If you get stuck on something, asking for help is one way to learn. But don't hesitate to learn independently too! There are many online tutorials, videos, and books out there t
Ln 9, Col 1 Spaces: 4 UTF-8 LF Python 3.11.2 ('activate': ven
raspberrypi.local (WayVNC) - RealVNC Viewer
audio_20241124_191515.wav - llm_folder - Visual Studio Code
LLM_FOLDER
activate
input_audio
audio_20241124_191341.wav
output_audio
audio_20241124_191515.wav
vectorstore.faiss
app.py
input.txt
output.txt
stt.py
tts.py
0:02 / 0:02
(activate) pi@raspberrypi:~/llm_folder $ python3 tts.py
Enter the text to convert to speech: hello i am somesh and learning iot
You entered: hello i am somesh and learning iot
Text appended to output.txt
Generated TTS file: output_audio/audio_20241124_191515.wav
(activate) pi@raspberrypi:~/llm_folder $


raspberrypi.local (WayVNC) - RealVNC Viewer
pi@raspberrypi: ~/ll..
audio_20241124_19..
19:13
audio_20241124_191341.wav - llm_folder - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
LLM_FOLDER
activate
input_audio
audio_20241124_191341.wav
vectorstore faiss
app.py
input.txt
stt.py
tts.py
stt.py
audio_20241124_191341.wav
tts.py
input_audio > audio_20241124_191341.wav
0:00 / 0:04
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
ALSA lib confmisc.c:160:(snd_config_get_card) Invalid field card
ALSA lib pcm_usb_stream.c:482:(_snd_pcm_usb_stream_open) Invalid card 'card'
ALSA lib confmisc.c:160:(snd_config_get_card) Invalid field card
ALSA lib pcm_usb_stream.c:482:(_snd_pcm_usb_stream_open) Invalid card 'card'
ALSA lib pcm_dmix.c:999:(snd_pcm_dmix_open) unable to open slave
Cannot connect to server socket err = No such file or directory
Cannot connect to server request channel
jack server is not running or cannot be started
JackShmReadWritePtr::~JackShmReadWritePtr - Init not done for -1, skipping unlock
JackShmReadWritePtr::~JackShmReadWritePtr - Init not done for -1, skipping unlock
Adjusting for ambient noise... please wait
Listening for speech...
Recording audio...
Processing audio...
You said: hello my name is Somesh Ghosh and I am learning embedded
Audio saved to input_audio/audio_20241124_191341.wav
Text appended to input.txt
Listening process ended.
(activate) pi@raspberrypi:~/llm_folder $
bash
python
bash
OUTLINE
TIMELINE
404.04KB