Avatar AI Assistant:

The Project Overview (Till 1st Month progress)

1. Environment Setup Procedure: (More detail on this link)

In order to run the “Avatar AI Assistant” system, the following library and tools are required:

a. ***Programming language*** **:**
   i. Python
b. ***Python dependencies:***
   The following should be installed for the system to work.
   i. SpeechRecognition==3.10.3
   ii. openai==1.16.2
   iii. pygame==2.5.2
   iv. pyaudio==0.2.14
   v. pyttsx3==2.90
   vi. python-osc==1.8.3
   vii. setuptools==69.5.1
   viii. fastapi==0.111.0
   ix. uvicorn==0.29.0
c. ***APP and Tools :***
   i. VR-Chat App and account to play the VRChat.
   ii. VB-Audio Cables (A+B)
   iii. OpeanAI Api key

Installation
For installation and configuration please refer to this file:
https://docs.google.com/document/d/1zEuwX3yFymS59yv9CwkLvNGnbsEIQe42Os3aVKNYJN0/edit#heading=h.vku93wir2pfa

2. VR Chat World Chat UI Creation (Unity) :

Please refer to this doc for VRChat World “Chat UI and history display UI” creation in creator companion. (link for Avatar AI Text Chat UI Creation)

3. Running the Project (AI Avatar Speech Chat + Text Chat UI)

➢ Once the VRChat world with Chat UI had been created and uploaded to VRChat (can follow this link to create a similar world) , we can then proceed to test the both Speech as well as Text chat functionalities in that world by following the steps outlined below: (Please note that we require two devices to test the system, and the requirements mentioned above in the Environment Setup Procedure are already installed or fulfilled to make the system work.)

1. Login and Enter VRChat World:
    - Begin by logging in with the AI Avatar account in the VRChat app on the first PC.
    - Enter the world that we created with that AI Avatar account.
2. Open Terminals and Navigate to Project Folder:
    - Open two command terminals on the first PC (where the VRChat app has AI Avatar logged in).
    - Navigate to the project folder in both terminals.
3. Install Python Packages:
    - In any opened command terminal that has been navigated to the project folder, execute the following command:
        - pip install -r requirements.txt
4. Run FastAPI File:
    - Type the following command to run the fastAPI file in the first command terminal:
        - uvicorn fastapi_vr_textchat:app --port 8000 --reload
        - This creates a backend server for facilitating the text chat, text chat session creation, and chat history data creation and extraction for VRChat.
5. Run Speech Chat File:
    - Similarly, type the following command to run the run.py file in the second command terminal:
        - python run.py
        - This enables the speech chat to be performed from the VRChat app and other features such as OSC messaging for speech box, facial expression display, etc.
6. Launch VRChat on Second PC:
    - On the second PC, launch the VRChat app and log in with any test user account.
    - Enter the same world where the AI Avatar is present.
7. Initiate Interaction:
    - Upon entering the ChatUI world with the test account, you will encounter the AI Avatar and a Chat UI panel.
    - Address the AI Avatar by saying "hello". It will awaken and engage in conversation with you.
    - Perform Speech chatting with the AI.

- Similarly, you can initiate a text chat by clicking the “Start new Chat” button (Unity 3d Cube) and then start typing a text message into the input field.
- Note that in the current implementation, the message format should adhere to something like this:: https://my_fastapi_back/message/Your_actual_message_here. (this is because we have used VRCURLInputField to take a message that needs a URL to be typed in the field.)

8. Creating New Sessions:
    - For speech chat: Say “Start new session” to AI avatar, and the next conversation onwards will be given a new session Id until another “Start new session” utterance is not heard.
    - For text chat: Users can create a new session for the chat by clicking the “start new chat” cube button when they want a new chat session.
9. Viewing Chat History:
    - The chat history can be visualized via the TMXPro text output field in the VRChat when the history button (Unity 3d Cube) is clicked. This will show the history based on sessions to the User.

4. Overview of the System (Working Flow and Technical Detail) :

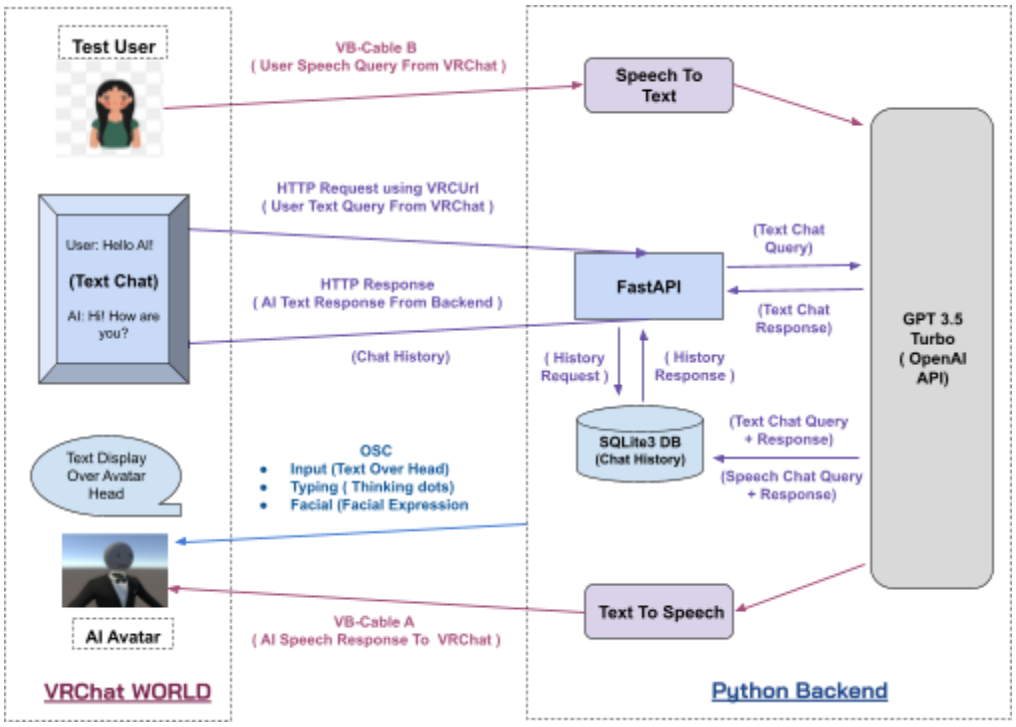


Fig 1: Overall working diagram

The module consist of Mainly Two components:

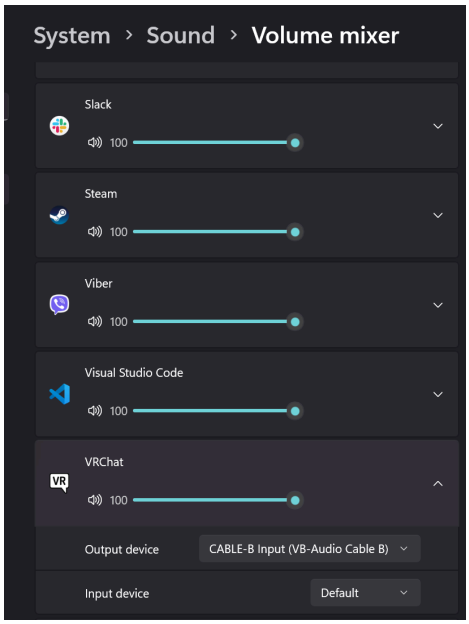
1. Speech Conversation
2. Text Conversation

Additional components:

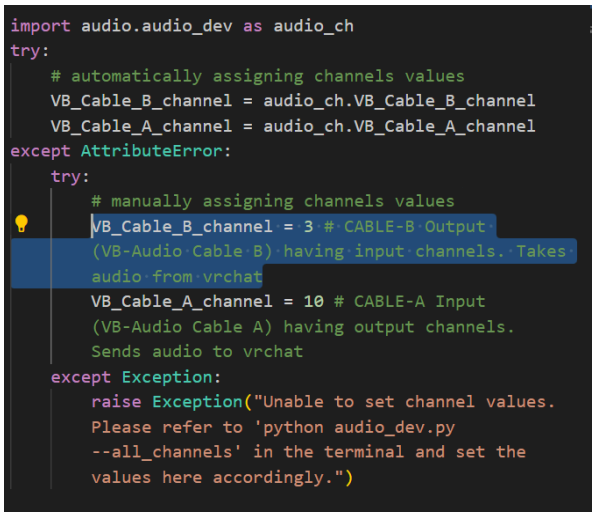
1. Text Showing over head
2. Emotion and expression
3. Chat History

A. Speech Conversation:

- ➢ This system takes the speech query from VRChat and provides AI responses in the form of Speech back to the user via an AI Avatar.
- ➢ The Speech Conversation system is activated when a user says the wake word “hello” in the VRChat World near the AI Avatar.
- ➢ For this the system consistently listens for the wake word and once it is heard, the AI Speech conversation starts and users can talk to the AI in any topic they like.
  a. The Speech conversation system consist of following steps:
    i. Step1: Capturing the speech of User from VRChat:
      - The VRChat audio output (User’s speaking) is first taken out to our Python code running in our PC using the “VB Audio Cable B”.
      - For this, go to “ Volume mixer” settings in Windows and choose the output device of the VRChat to “CABLE - B Input (VB Audio Cable B)”



- SImilarly, in the python script the “CABLE - B Output (VB Audio Cable B) “ should be the corresponding settings for receiving the audio from VRChat. In fact, the project already has a script “audio_dev” that automatically handles task of assigning the Cables in python script.



```
import audio.audio_dev as audio_ch
try:
    # automatically assigning channels values
    VB_Cable_B_channel = audio_ch.VB_Cable_B_channel
    VB_Cable_A_channel = audio_ch.VB_Cable_A_channel
except AttributeError:
    try:
        # manually assigning channels values
        VB_Cable_B_channel = 3 # CABLE-B Output
        (VB-Audio Cable B) having input channels. Takes
        audio from vrchat
        VB_Cable_A_channel = 10 # CABLE-A Input
        (VB-Audio Cable A) having output channels.
        Sends audio to vrchat
    except Exception:
        raise Exception("Unable to set channel values.
        Please refer to 'python audio_dev.py
        --all_channels' in the terminal and set the
        values here accordingly.")
```

ii. Step 2: Conversion of User’s speech captured from VRChat to text in python:
- The user speech taken from the CABLE-B Output (VB Audio Cable) in python, is then converted to text using the “SpeechRecognition” module.

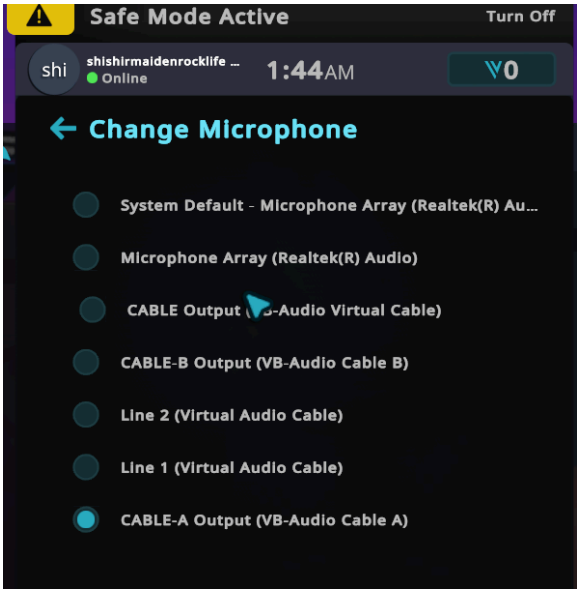
iii. Step 3: AI Response Generation:
- The text (transcribed user speech) is then passed to “GPT 3.5 turbo using OpenAI Api) which generates the response for the given text.

iv. Step 4: Text To Speech Conversion:
- The response from the GPT 3.5 is then converted to speech using the “Pyttsx3” text to speech module.

v. Step 5: Feeding the response speech to VRChat:
- The speech response from GPT 3.5 (AI) is then streamed to VRChat App where the AI Avatar utters the speech.
- For transferring the audio from the python backend to VRChat, we use the “VB Audio Cable A”.
- For this, we need to set the “Output device index” in python file to “Cable-A Input (VB-Audio Cable A)”

```
import audio.audio_dev as audio_ch
try:
    # automatically assigning channels values
    VB_Cable_B_channel = audio_ch.VB_Cable_B_channel
    VB_Cable_A_channel = audio_ch.VB_Cable_A_channel
except AttributeError:
    try:
        # manually assigning channels values
        VB_Cable_B_channel = 3 # CABLE-B Output
        (VB-Audio Cable B) having input channels. Takes
        audio from vrchat
        VB_Cable_A_channel = 10 # CABLE-A Input
        (VB-Audio Cable A) having output channels.
        Sends audio to vrchat
    except Exception:
        raise Exception("Unable to set channel values.
        Please refer to 'python audio_dev.py
        --all_channels' in the terminal and set the
        values here accordingly.")
```

- And, in the VRChat app, The Microphone setting should be set to “CABLE-A Output (VB_Audio Cable A)”



B. Text Conversation:

- ➢ The Text conversation module lets VRChat’s users communicate with AI using Text in the Text chat UI. For this the user’s are provided with a Text Chat UI in the VRChat world we particularly created using Unity and Udon Sharp.
- ➢ The overall essential steps for creating the VRChat World with Text Chat UI and sending/handling of the users/AI message using FastAPI and UdonSharp is defined in this document “Avatar_AI Text Chat UI Creation”.

a. The overall flow of Text conversation with the AI using the Text Chat UI in the VRChat world consist of following steps:
    i. User query message input:
        - The User can type in a message in the UI Text Field in our VRChat World
        - Right now to pass the message from VRChat (UdonSharp) to the Python backend, we have used VRCUrlInputField and string loader method. VRCUrlInputField allows typing of URL (backend endpoint) during the runtime. And the String Loder method allows us to make HTTP requests from Udon Sharp, to our backend endpoint URL along with the user message query attached to the URL itself.
        - Thus, the query should be written in the from “URL/messege/users actual message”. (for e.g: https://tame-jokes-repair.loca.lt/message/Hello MR AI) where, “Hello MR AI is actual message”.
    ii. AI response generation:
        - The user’s text query message sent using VRCUrlInputField is received by the FastAPI endpoint (ie: `@app.get("/message/{message}")`).
        - The user’s text query message is then passed to GPT 3.5 (Openai API), which generates the response to the user’s query text.
    iii. Transfer of AI response from FastAPI server to VRChat.
        - The AI response along with the corresponding user’s query is passed to VRChat as a response to the same HTTP request made earlier to pass the user’s query to Backend.
    iv. Displaying User query and AI response in the Chat Canvas.
        - The user query and AI response thus passed back to the VRChat is in actual received by the UdonSharp script that leverages the “string-loading” functions (“VRCStringDownloader” and “IVRCStringDownload“, to download the response json string (User query + AI message) and decode via UTF8 standard respectively)
        - Then the result ie (User query and AI message is then displayed in the VR chat using the UI Canvas through the “TextMeshPro text”.

          Reference to String Loading : https://creators.vrchat.com/worlds/udon/string-loading/

C. Text Showing overhead with a thinking expression.

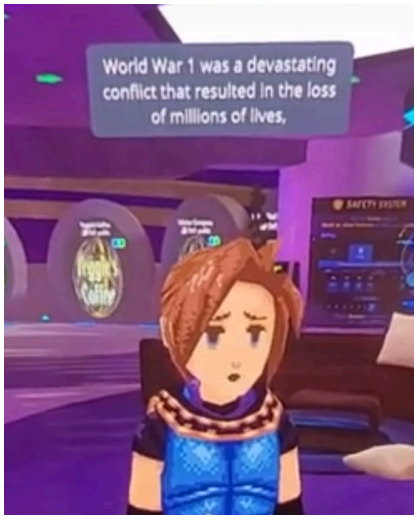
➢ In order to display the AI response over the head of the avatar, the “OSC as Input Controller” in VRChat is leveraged. These responses are shown within a chatbox positioned just above the avatar, creating a lifelike dialogue experience.

➢ Additionally, alongside transmitting AI responses through methods like VB cable for speech and string loading method to make Http request/response for text, we leverage the OSC method as well. This method grants us the ability to manipulate VRChat's default keyboard behavior.

➢ Text sent via 'python-osc' seamlessly integrates into VRChat, appearing as if typed by a user. Consequently, AI-generated text is displayed above the avatar's head, resembling a speech bubble. This versatile approach enables the display of AI responses from any module, whether originating from speech conversations or text chats

The Text showing over the head of the avatar consist of the following steps:

- The server address and server port is assigned as " ***server_address = '127.0.0.1' and server_port = 9000".***
- A UDP client is setup using "udp_client.SImpleUDPClient method from pythonosc library". (ie: ***client = udp_client.SimpleUDPClient(server_address, server_port***). (Reference for SimpleUDPClient https://python-osc.readthedocs.io/en/latest/client.html)
- WIth this setup, the AI response generated using the GPT 3.5 is sent to VRChat using the OSC method through the "Chatbox" parameter as (having ***osc_address = '/chatbox/input'***
  and ***osc value = AI generated text.***
- Upon being sent, the OSC message (ie: GPT 3.5 generated text response) is shown above the avatar head.
    - In the project, I have set the sending of text to match the speed of speech by sending only at most 16 words a time that last in the display for 6 seconds over the avatar's head.)



- Further, to enhance the speech box shown above the Avatar, an additional "thinking dot" is also reflected when the avatar is waiting for the response from backend. For this, the '/chatbox/typing' OSC is used. To achieve this, The '/chatbox/typing/' is kept active until the response from the backend is received.

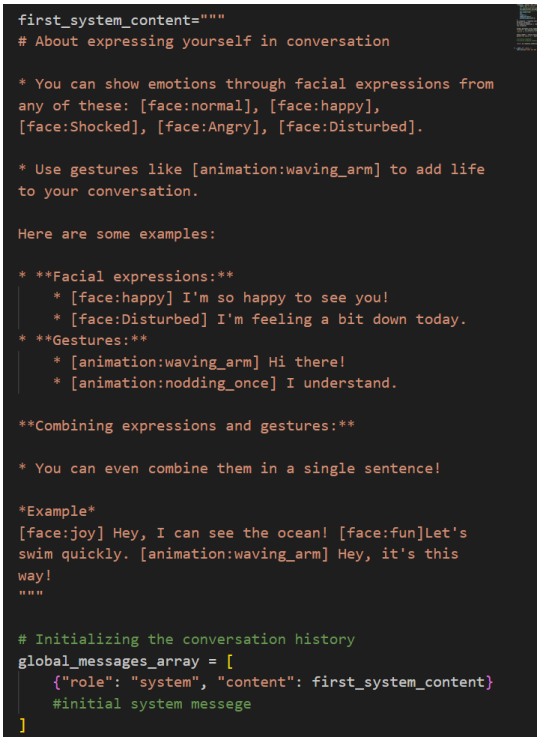
D. Emotion and expression:

➢ The AI Avatar is designed to convey emotions and expressions based on the responses it generates to address the queries of VRChat users.

- ➢ To enable the Avatar to display these emotions, it must be equipped with appropriate emotional and expressive components, which can be created using tools like Blender or Unity.
- ➢ However, currently, we are utilizing a prebuilt Avatar capable of showcasing facial expressions name “Alex”

Given that we are using a prebuilt Avatar with expression features, our focus shifts to two things:

a. Effectively extracting emotions from the response text generated by the GPT model:
   i. In this scenario, the GPT (OpenAI API) itself can be leveraged to assign or extract the emotion from the answer text it gives.
      - This involves setting up a prompt in such a way that with each GPT response, the model provides the most relevant emotion value from a predefined set.
      - For instance, we establish a prompt structure in a way that it will ask the GPT model not just for providing a response to the user query but also to include the emotion within the generated responses.



```
first_system_content="""
# About expressing yourself in conversation

* You can show emotions through facial expressions from
any of these: [face:normal], [face:happy],
[face:Shocked], [face:Angry], [face:Disturbed].

* Use gestures like [animation:waving_arm] to add life
to your conversation.

Here are some examples:

* **Facial expressions:**
    * [face:happy] I'm so happy to see you!
    * [face:Disturbed] I'm feeling a bit down today.
* **Gestures:**
    * [animation:waving_arm] Hi there!
    * [animation:nodding_once] I understand.

**Combining expressions and gestures:**

* You can even combine them in a single sentence!

*Example*
[face:joy] Hey, I can see the ocean! [face:fun]Let's
swim quickly. [animation:waving_arm] Hey, it's this
way!
"""

# Initializing the conversation history
global_messages_array = [
    {"role": "system", "content": first_system_content}
    #initial system messege
]
```

   ii. As a result the output from the GPT will be like this: “*[face:joy] I am glad you are celebrating your birthday. Enjoy your day to the fullest. “*

```python
import re

emotions_dict={
    '[face:normal]':1,
    '[face:happy]':2,
    '[face:Shocked]':3,
    '[face:Angry]':4,
    '[face:Disturbed]':5
}

def check_expression(ai_response_emotion):
    emotion_animation_regex = r"\[(face:\w+)\]|\[animation:\w+\]"  # Combined pattern
    ## tracking the emotion
    match = re.search(emotion_animation_regex, ai_response_emotion)
    # If a match is found, extract the emotion
    if match:
        extracted_emotion = match.group(0)
        # print(f"Extracted emotion: {extracted_emotion}")
        if extracted_emotion in emotions_dict:
            emotion_number = emotions_dict[extracted_emotion]
            # print("emotion_value:",emotion_number)

            return emotion_number
    else:
        default_emote = 1
        return default_emote
```

b. Conveying the emotion for the generated AI responses to VRChat for displaying facial expression the AI avatar's face when it utters the response speech.

i. Now, we can apply a regex check to extract the emotion aspect placed by GPT for the response.

Ii. We create a index mapping to represent each emotion with a number using a python dictionary

Iii Subsequently, these emotion numbers will be passed to the VRChat via OSC utilizing the Face/Facial OSC parameter (for this the Avatar should have a Face/Facial parameter in it or we can create the parameter using Unity or Blender)

- To perform the OSC transmission, we set up the udp_client.SimpleUdpClient to '127.0.0.1', 9000 which is the default url and port for OSC in VRchat.
- The values of the emotion (index number) in the response is set as the osc_value and the `'/avatar/parameters/Face/Facial'` is set as the OSC_address.
- Now upon sending the OSC message with the above osc_adress and values as follows which will result in the avatar to show the emotion in VRChat.

```python
def facial_express(emo_val):
    # Define the OSC address and message to send
    osc_address = '/avatar/parameters/Face/Facial'
    osc_value = emo_val

    # Send the OSC message
    client.send_message(osc_address, osc_value)
```

E. Chat History

i. The Chat History is maintained using SQLite3 database.
ii Each user query and response are saved in the database based on the sessions.

- A new sessions for Speech is created in two scenarios:
    - When the user says hello as the first wake word.
    - Once, the wake word is detected and conversation starts, the user can create new session by saying “Start New Session”
    - Subsequently, the entire conversation within the single session are saved in the database with the same “session id”.
- Likewise, a new session for Text chat is created when a user clicks on the “Start New Session” Tmx cube in our VRChat World Text chat UI.
    - Upon clicking the “Start New Chat”, an empty http request is sent to the url set in the VRCUrl ie: our FastApi endpoint responsible for assigning a new session id for the conversation.
    - The FastAPI endpoint will handle the request and create a new session in the backend. “`@app.get("/newsession")`”
    - Similar to speech conversation data saving the whole conversation within a single session is kept with the same session id in the database.
- Thus, the database will contain the following information:
    - Session id
    - Source : The source from where the conversation is getting logged from (ie: either text or speech conversation)
    - Message : the user message
    - Response : the AI Response to the user’s message
    - Timestamp : Keep the time of the instance when the chat is happening.

iii. Excessing the Chat History in the server (SQLite3 database) from VRChat:

- A fastAPI endpoint `@app.get("/chat_history"` is established in the project to which the VRC will communicate using the HTTP request to get chat history data as response
- To achieve this, VRCUrl in the UdonScript is set to the url to our fast api endpoint responsible for processing the chat history request. Likewise, VRCStringDownloader and IVRStringDownload are leveraged to download and display the conversation history from fast api endpoint as string.
- When the “Chat history” button is clicked in the text chat UI in the world, this fastAPI endpoint receives a GET request. The fastAPI endpoint then executes the function that queries the database for all the conversation history sorted based on session.

- The data is extracted from the database and this fastAPI endpoint passes the data in the from of json back to VRChat as a response to the previous request call.
- The json data is received using the VRCString Downloader (VRCStringDownloader and IVRStringDownload.) in the UdonSharp script.
- Finally, the history is displayed in the scroll view in the VRChat using TextMesh Pro.

5. Errors encountered and Important References are in this document link : [Link to Avatar AI Error]