Video Call Application (WebRTC) with audio driven talking head generation

This the first iteration, where we are focused on installing python 3.X, setting up conda environment, installing required libraries for the project execution. Apart from that we experimented and learned about WebRTC + Streamlit. The decision is made on the ML model to implement the chosen audio driven approach.

Installing python 3.X:

All the models were implemented using Python 3.6, hence as to implement the same models we decided on installing python 3.6

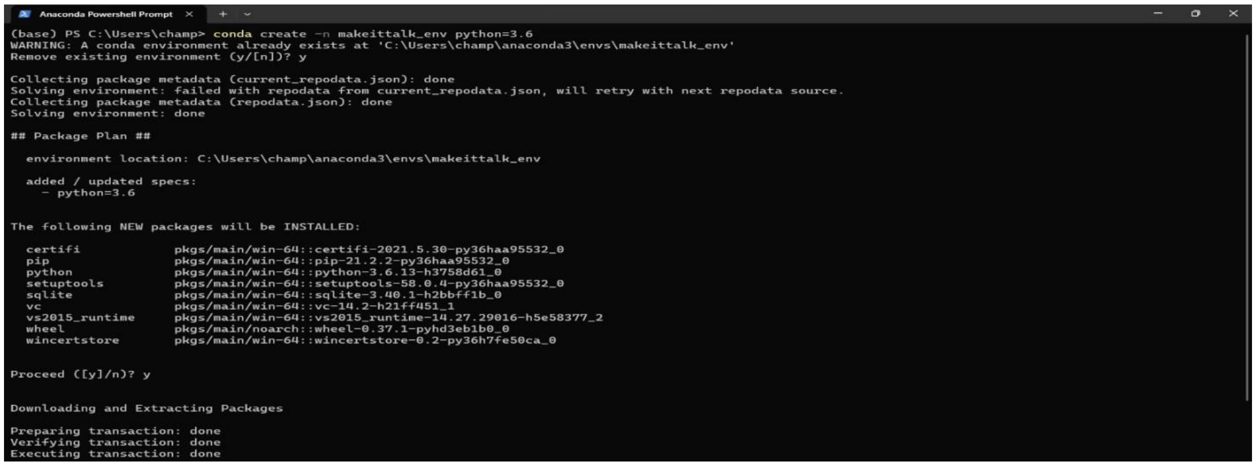
The list of libraries needed to implement:

- ffmpeg-python: This is used to trim, concatenate and overlay the audio files that we record.its command-line interface.
- opencv-python: This can be utilized when dealing with image processing. In our case this library comes handy when we overlap caller's face with talking head image.
- face_alignment: It is used to detect 2D and 3D face alignment.
- scikit-learn: library for machine learning in Python. This provides us a selection of efficient tools for machine learning where we can implement regression, clustering and dimensionality reduction via a consistence interface.
- pydub: This library is used to handle the .wav files which are generated after recording the audio.
- pynormalize: Using this we save the metadata of the original audio file,which can be used for processing.
- soundfile: Read and write operations of audio file are performed using this module.
- librosa: This is used to normalize the audio file and to extract features of audio file.
- pysptk: It's a python wrapper library for speech signal processing.
- pyworld: This is used for speech breakdown in our project.
- resemblyzer: It is utilized to implement deep learning part of our project.

Setting up conda environment:

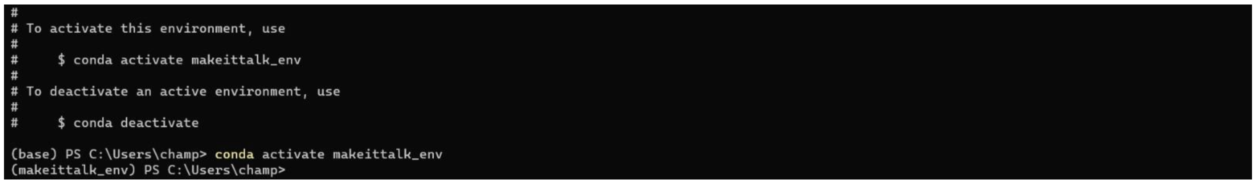
We created a new environment in conda and installed the above-mentioned dependencies(libraries) to further execute ML model. There are basically two steps involved one creation and other activation.

Makeittalk is the new environment that we tried to create.



```
(base) PS C:\Users\champ> conda create -n makeittalk_env python=3.6
WARNING: A conda environment already exists at 'C:\Users\champ\anaconda3\envs\makeittalk_env'
Remove existing environment (y/[n])? y

Collecting package metadata (current_repodata.json): done
Solving environment: failed with repodata from current_repodata.json, will retry with next repodata source.
Collecting package metadata (repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: C:\Users\champ\anaconda3\envs\makeittalk_env

  added / updated specs:
    - python=3.6


The following NEW packages will be INSTALLED:

  certifi            pkgs/main/win-64::certifi-2021.5.30-py36haa95532_0
  pip                pkgs/main/win-64::pip-21.2.2-py36haa95532_0
  python             pkgs/main/win-64::python-3.6.13-h3758d61_0
  setuptools         pkgs/main/win-64::setuptools-58.0.4-py36haa95532_0
  sqlite             pkgs/main/win-64::sqlite-3.40.1-h2bbff1b_0
  vc                 pkgs/main/win-64::vc-14.2-h21ff451_1
  vs2015_runtime     pkgs/main/win-64::vs2015_runtime-14.27.29016-h5e58377_2
  wheel              pkgs/main/noarch::wheel-0.37.1-pyhd3eb1b0_0
  wincertstore       pkgs/main/win-64::wincertstore-0.2-py36h7fe50ca_0


Proceed ([y]/n)? y


Downloading and Extracting Packages

Preparing transaction: done
Verifying transaction: done
Executing transaction: done
```

Fig: Makeittalk environment is created



```
#
# To activate this environment, use
#
#     $ conda activate makeittalk_env
#
# To deactivate an active environment, use
#
#     $ conda deactivate

(base) PS C:\Users\champ> conda activate makeittalk_env
(makeittalk_env) PS C:\Users\champ>
```

Fig: Environment activation

WebRTC+Streamlit:

We are using Streamlit to develop the UI of our project and WebRTC is used to provide the real-time video communication. The two libraries for this purpose are streamlit and stream-webrtc.

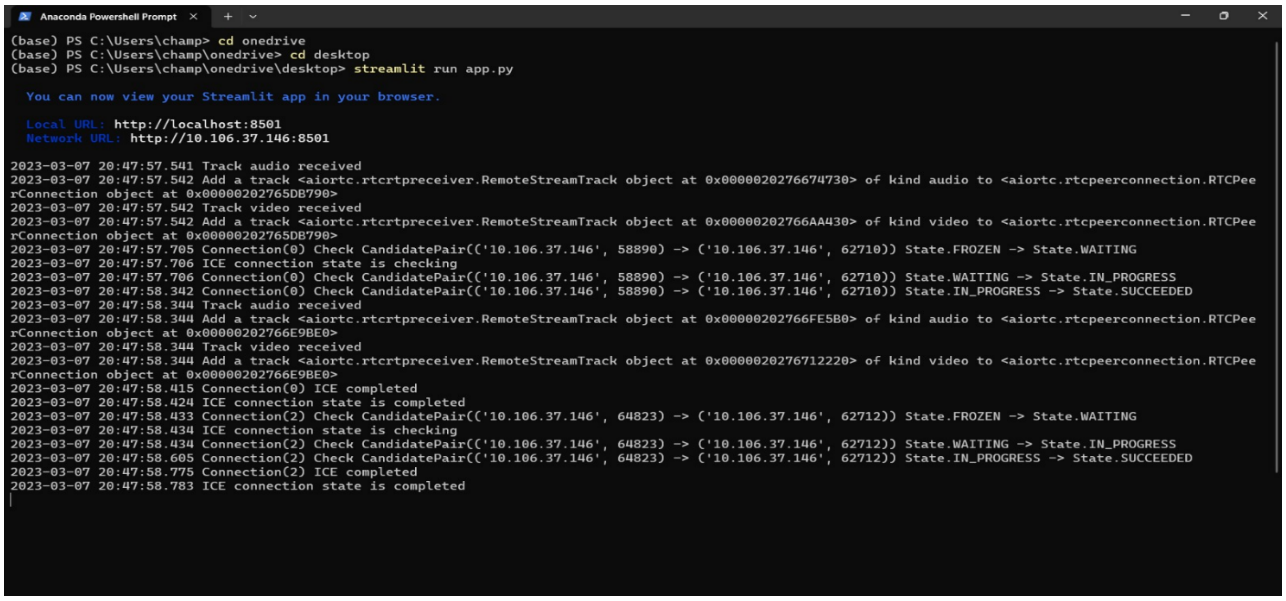
The commands to install python libraries required to have this setup are:

- Pip install Streamlit
- Pip install streamlit-webrtc

After studying about streamlit and webrtc protocol, we locally tried multiple bits and pieces of code to get hold of the structure and development environment. Finally we were able to develop a streamlit webpage that consists of two windows which are enabled with camera and microphone, and these can in future be modified to create a real-time call scenario. The webpage also consists of an option where user can chose the talking head.

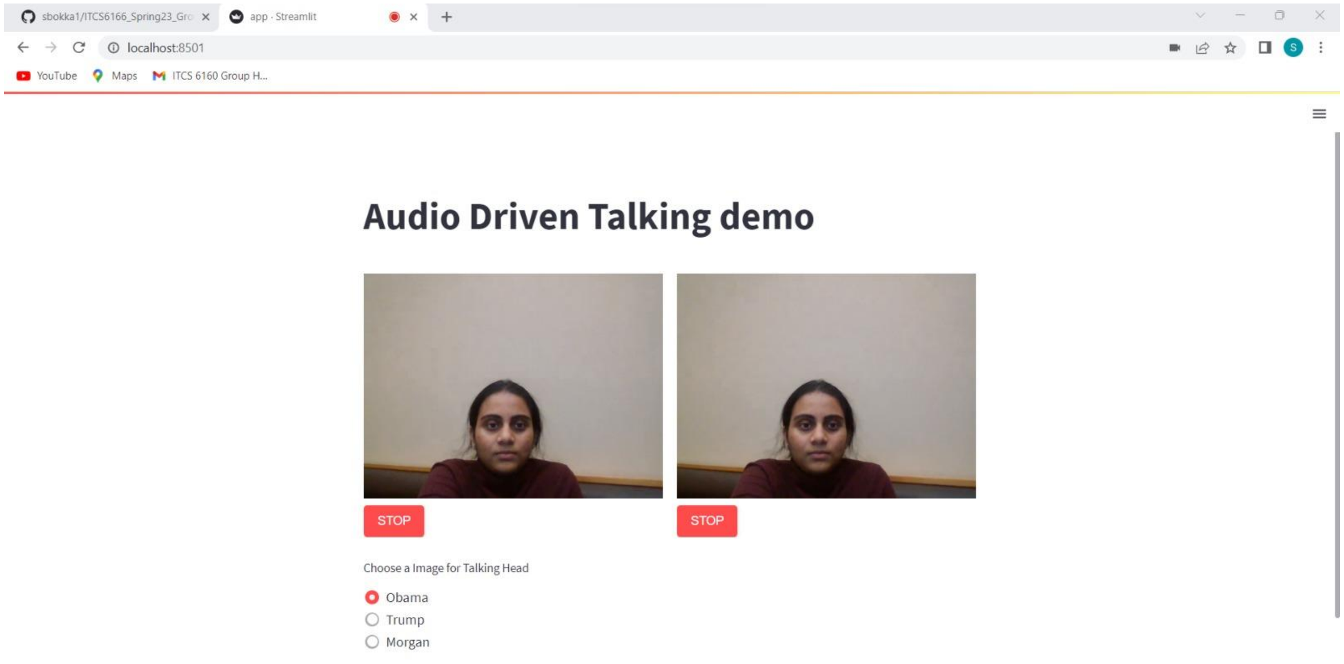
Instructions to run the application:

- Navigate to the folder where the application resides.
- To run the application, use the syntax: “streamlit run filename.py”



```
(base) PS C:\Users\champ> cd onedrive
(base) PS C:\Users\champ\onedrive> cd desktop
(base) PS C:\Users\champ\onedrive\desktop> streamlit run app.py

  You can now view your Streamlit app in your browser.

  Local URL: http://localhost:8501
  Network URL: http://10.106.37.146:8501

2023-03-07 20:47:57.541 Track audio received
2023-03-07 20:47:57.542 Add a track <aiortc.rtcrtpreceiver.RemoteStreamTrack object at 0x0000020276674730> of kind audio to <aiortc.rtcpeerconnection.RTCPee
rConnection object at 0x00000202765DB790>
2023-03-07 20:47:57.542 Track video received
2023-03-07 20:47:57.542 Add a track <aiortc.rtcrtpreceiver.RemoteStreamTrack object at 0x00000202766AA430> of kind video to <aiortc.rtcpeerconnection.RTCPee
rConnection object at 0x00000202765DB790>
2023-03-07 20:47:57.705 Connection(0) Check CandidatePair(('10.106.37.146', 58890) -> ('10.106.37.146', 62710)) State.FROZEN -> State.WAITING
2023-03-07 20:47:57.706 ICE connection state is checking
2023-03-07 20:47:57.706 Connection(0) Check CandidatePair(('10.106.37.146', 58890) -> ('10.106.37.146', 62710)) State.WAITING -> State.IN_PROGRESS
2023-03-07 20:47:58.342 Connection(0) Check CandidatePair(('10.106.37.146', 58890) -> ('10.106.37.146', 62710)) State.IN_PROGRESS -> State.SUCCEEDED
2023-03-07 20:47:58.344 Track audio received
2023-03-07 20:47:58.344 Add a track <aiortc.rtcrtpreceiver.RemoteStreamTrack object at 0x00000202766FE5B0> of kind audio to <aiortc.rtcpeerconnection.RTCPee
rConnection object at 0x00000202766E9BE0>
2023-03-07 20:47:58.344 Track video received
2023-03-07 20:47:58.344 Add a track <aiortc.rtcrtpreceiver.RemoteStreamTrack object at 0x0000020276712220> of kind video to <aiortc.rtcpeerconnection.RTCPee
rConnection object at 0x00000202766E9BE0>
2023-03-07 20:47:58.415 Connection(0) ICE completed
2023-03-07 20:47:58.424 ICE connection state is completed
2023-03-07 20:47:58.433 Connection(2) Check CandidatePair(('10.106.37.146', 64823) -> ('10.106.37.146', 62712)) State.FROZEN -> State.WAITING
2023-03-07 20:47:58.434 ICE connection state is checking
2023-03-07 20:47:58.434 Connection(2) Check CandidatePair(('10.106.37.146', 64823) -> ('10.106.37.146', 62712)) State.WAITING -> State.IN_PROGRESS
2023-03-07 20:47:58.605 Connection(2) Check CandidatePair(('10.106.37.146', 64823) -> ('10.106.37.146', 62712)) State.IN_PROGRESS -> State.SUCCEEDED
2023-03-07 20:47:58.775 Connection(2) ICE completed
2023-03-07 20:47:58.783 ICE connection state is completed
```

Fig: Screenshot representing the run instructions and run results.



Audio Driven Talking demo

Choose a Image for Talking Head

[✓ Obama]
Trump
Morgan

Fig: Streamlit page

The above shown image is our initial attempt towards the UI of the project and it will be further developed in coming iterations. The code file corresponding to the above page is attached in the GitHub.

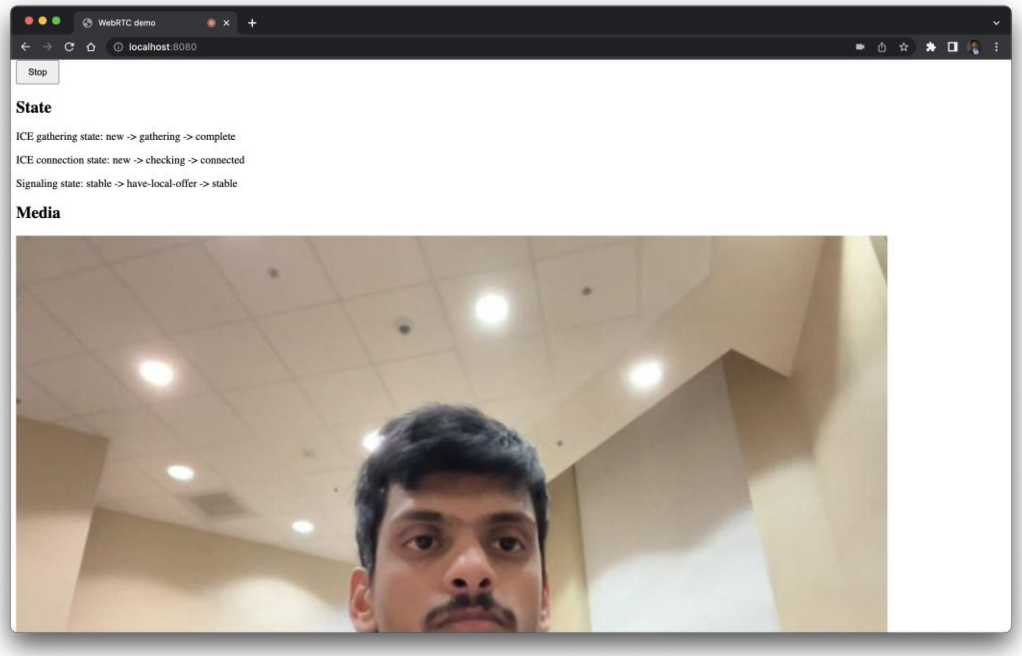
Decision on the Model:

After closely examining both models i.e Makeittalk and livespeech portraits. We finally decided on proceeding with Makeittalk model. For the following reasons: Both models are almost similar. Makeittalk model is additionally using libraries like face_alignment to get the precise dimensions of the user face in-order to ensure perfect overlap. The makeittalk model also utilizes libraries that can better process the audio files when compared to libraries in livespeech portraits.

aioRTC Experiment:

We used aiortc repo: https://github.com/aiortc/aiortc to learn to understand how WebRTC works and tinker with its internals. Since the repository is quite old, newer version of python didn't work. So I figured python3.9 worked well.

We used the server example from the above repository. This example illustrates establishing audio, video and a data channel with a browser. It also performs some image processing on the video frames using OpenCV.



State

ICE gathering state: new -> gathering -> complete

ICE connection state: new -> checking -> connected

Signaling state: stable -> have-local-offer -> stable

Media

Fig: Server example

We completed all the planned tasks of iteration-1 and besides that we tried to test the ML models, there were few bottle necks in testing. We are positive about overcoming them in further iterations.

The test results and the codes of implementation of webrtc , streamlit and ML model executions can be found in the respective individual folders.