Face Vitals V2

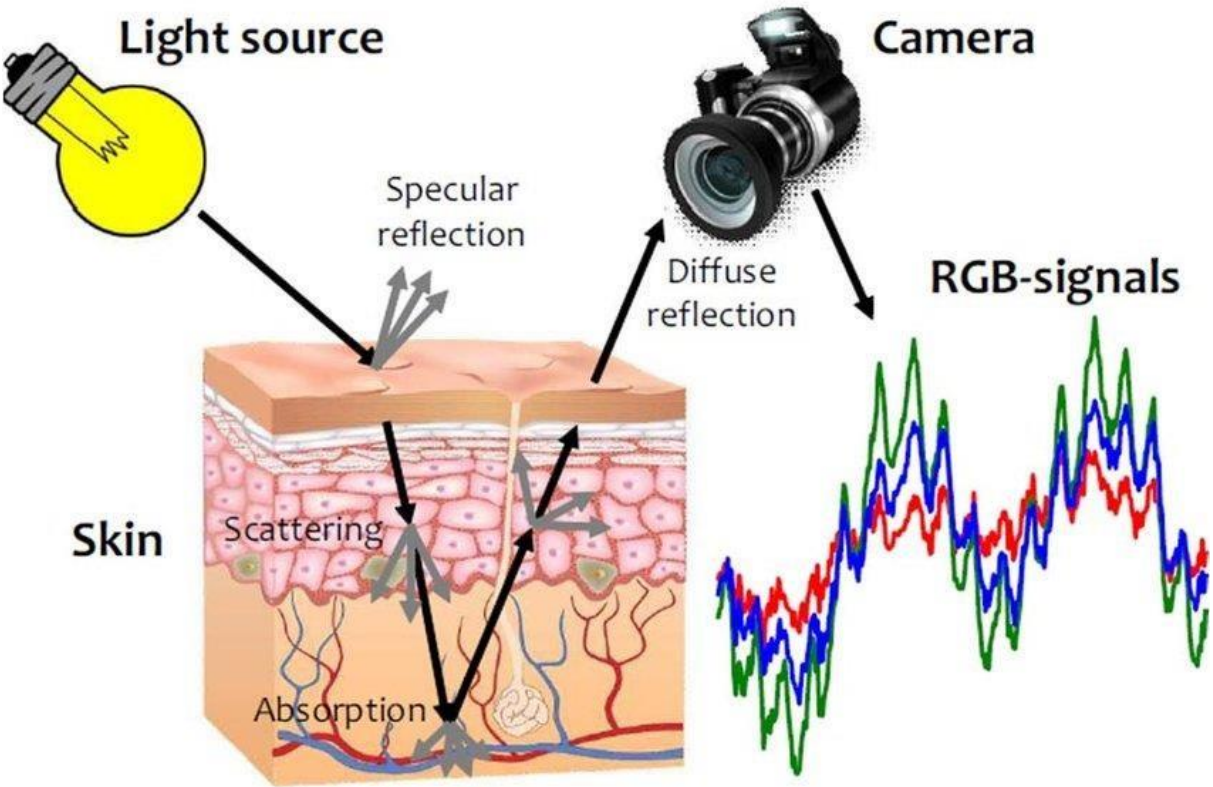
Objective

Given a small chunk of webcam video, predict the face vitals like

- Gender
- Age
- BMI (Body Mass Index)
- Heart Rate
- Respiration Rate
- Oxygen Saturation (SpO2) Level
- Stress/ No Stress

Remote Photoplethysmography (rPPG)

Remote Photoplethysmography (rPPG) is the backbone of this solution It is a non-invasive method used to measure vital signs, like heart rate, by analyzing minute changes in skin color caused by blood flow. These changes are captured through video footage, typically of the face, and processed using algorithms to extract the physiological signals. rPPG is particularly useful in scenarios where traditional contact-based monitoring is impractical or intrusive.



High Level Overview

- Front end: React JS and HTML, CSS (contact Rohit Jadhav)
- Back end: Flask Framework and Python
- The webcam video should be recorded at 30 fps and with a width of 640 and height of 480 for a duration of at least 10 seconds (or 300 frames)
- The front-end captures video for 15 seconds and sends it to backend endpoint "/save_video" that starts processing the POST request.
- The backend should always return some response, even if the processing fails, that is, there should be a response to display to end user regardless of the error or runtime errors.
- The ideal response time is below 10 seconds, however due to complexity and limited infrastructure it is taking around 50-60 seconds.

Key Assumptions

- **Ambient Lighting:** The video should be recorded in ambient lighting room. The rPPG process depends heavily on lighting as the main component is to analyse the light reflected of the facial region.
- **Melanin:** Melanin is the pigmentation caused in skin that gives it darker skin color. In general, the changes on skin are better on skins with lower melanin level. We have used a diverse dataset to overcome this problem. However, using quality dataset is recommended, specifically if it is being used to target a particular market.
- **Relaxed Position/ Movement of Subject:** The subject is the user whose webcam video will be analysed to predict face vitals. The model is robust enough to predict results with little movement, however a relaxed position results in much better prediction. The relaxed position is also preferred as the normal range for these vitals is given for relaxed position only. For example, respiration rate for adults in relaxed position is 12 to 20 BPM. A vigorous movement by default increases the respiration rate and result in erroneous predictions.

System Architecture

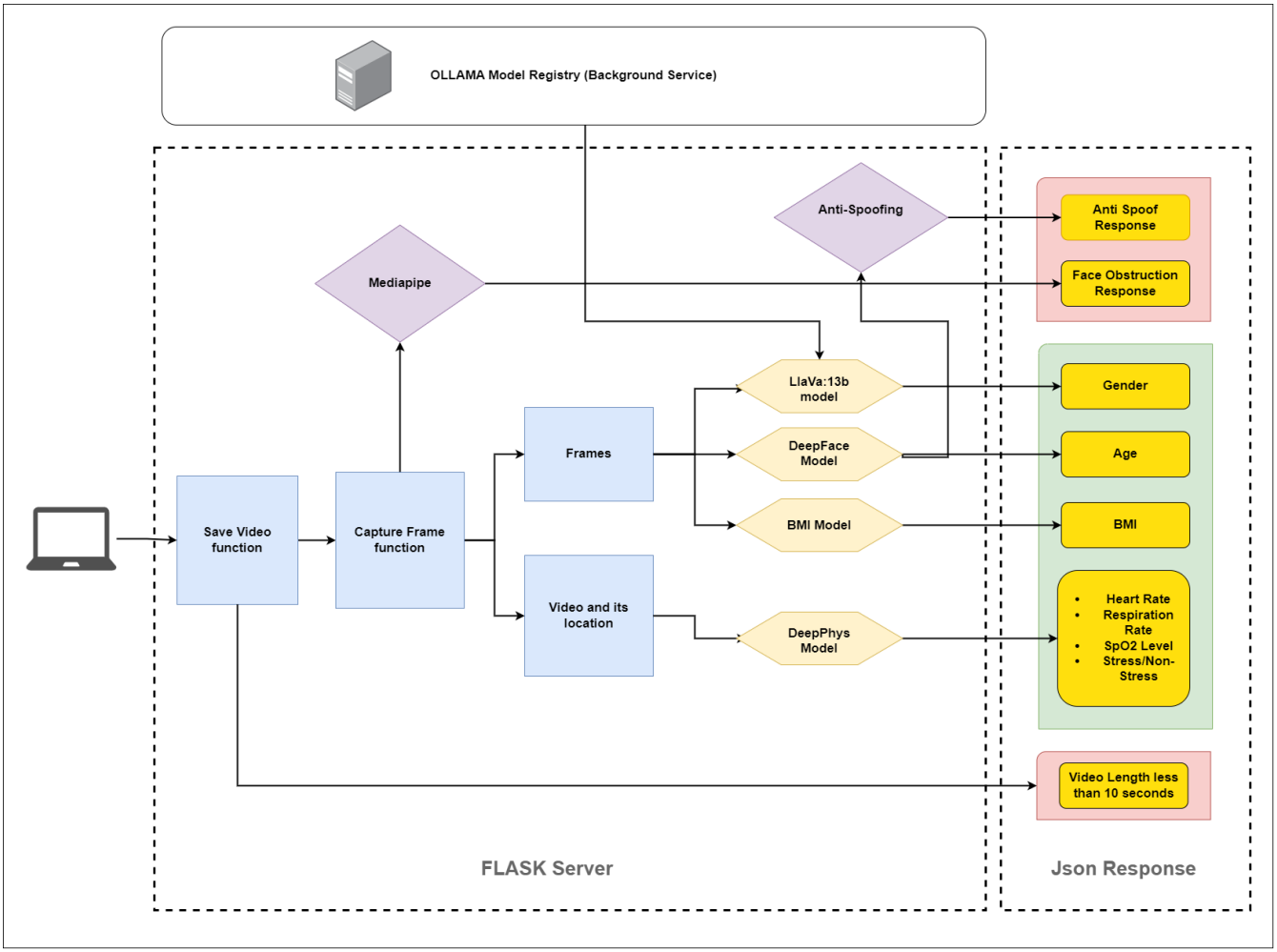
Technology Stack

- **Programming Language:** Python
- **Web Application Development Framework:** Flask
- **Front End Framework:** React JS
- **ML and DL Framework:** Tensorflow, PyTorch, Ollama
- **Deployment Server:** Linux, Nginx, Gunicorn, Linux commands
- **LLM:** llava:13b
- **CI/CD:** Github

Important Libraries

- Tensorflow==2.15.0
- PyTorch (ensure compatibility with the CUDA version)
- Flask
- OpenCV
- MediaPipe
- Deepface
- Torch and Torchvision
- Biosppy
- Numpy, Scipy
- Gunicorn

Architecture Diagram



Components Overview

1. **Face Obstruction Analysis:** The first task is to analyze if the face is obstructed in some way. Since we analyse the face region for predicting face vitals, it becomes paramount to ensure that the face is not obstructed. We use **MediaPipe** library to detect face and the confidence score. If the confidence score is less than 90%, we return an error response. The confidence score drops whenever the face is obstructed and has been tested manually with multiple objects as well as using one's own hands.

2. **Generating and extracting face region frames:** We use **MediaPipe** library to detect face for frame number 30, 60, 90, 120, 150, 180, 210, 240, and 270. Since video is recorded at 30 fps these frame number correspond to 1, 2, 3, 4, 5, 6,7, 8, and 9th second. The detected region is then expanded so that we have entire face frame saved rather than tightly bound face that only covers from forehead to chin. This is to ensure that age, gender and BMI model have proper context for prediction.

3. **Age Model:** We use **DeepFace** library to predict the age. The Age client is called to create model at runtime with an MTCNN detector for face recognition for age model. The predictions for all 9 frames are taken, and the **maximum** is used to predict the age. The **maximum** is taken to minimize the effect of erroneous predictions of a few frames. If we take the prediction for a single frame, the error prediction may be sent to the front end. Therefore, multiple frames are predicted, and the **maximum** is selected. The acceptable

range for age is plus or minus 2 years from the actual age. That is, if the user is 30 years of age, the desired predictions should be between 28 to 32.

4. **Gender Model:** Initially, we used the **DeepFace** library to predict gender. The Gender client was called to create the model at runtime. Predictions for all 9 frames were averaged to predict the gender. The average was taken to minimize the effect of erroneous predictions from a few frames. If we took the prediction from a single frame, an error could potentially be sent to the front end. Therefore, multiple frames were predicted and averaged out. The gender prediction was expected to always be correct. To overcome incorrect predictions, we suggested using an in-house model trained on an appropriate dataset with an expanded face region to accommodate more features, such as hairstyle.
   However, during evaluation it reveals notable limitations in model's gender prediction capabilities. While the model performs well on male datasets, its accuracy on female datasets is less consistent, leading to a bias toward male predictions. Factors such as makeup, poor lighting, low image quality, and females with traditionally masculine features frequently resulted in misclassifications. These issues highlight the model's struggle to differentiate between facial characters across genders, especially in less ideal conditions.
   To overcome these limitations, we came up with an alternate approach of using a **Vision-Language Model (VLM)**, specifically **LLaVA:13B**, to predict gender. VLMs combine visual and textual data, allowing them to understand images in a richer context. Instead of analyzing multiple frames, we now consider one random frame and provide additional textual prompts that offer more context alongside the image. This integration of image and text helps improve accuracy and reduces the complexity of working with multiple frames, as the model can focus on broader features like hairstyle, attire, and other visual cues.

5. **Anti-Spoofing Analysis:** The **DeepFace** library has anti-spoofing capabilities when it is used toextract faces. It is able to detect whenever a mobile phone is used in webcam. However, we need to ensure the version of DeepFace should be 0.0.21 and tensorflow==2.15.0. Currently,as observed the anti-spoofing works quite well in bright/natural lighting conditions. However,it creates wrong predictions in indoor lights. Therefore, we have currently kept the thresholdat 0.99.

6. **BMI Model:** The BMI model is build using Pytorch and Torchvision. It defines a custom model architecture by combining a pre-trained Vision Transformer (ViT) model with a custom head for a specific task, likely predicting BMI (Body Mass Index) from image data. The ViT model is pre-trained on the ImageNet dataset, specifically the SWAG variant which is fine-tuned for end-to-end tasks. The custom BMI Head created earlier is instantiated and assigned to model heads. This replaces the default head of the ViT model with the custom one designed for the BMI prediction task. The **Pre-trained ViT Model** extracts high-level features from images.
   Custom Head (BMI Head) processes these features through a series of linear layers and non-linear activations to output a single scalar value, intended to predict BMI. This setup is common in transfer learning, where the idea is to leverage a model pre-trained on a large dataset (like ImageNet) and adapt it to a specific task (like BMI prediction) by adding a custom head that is trained on the new task-specific data.

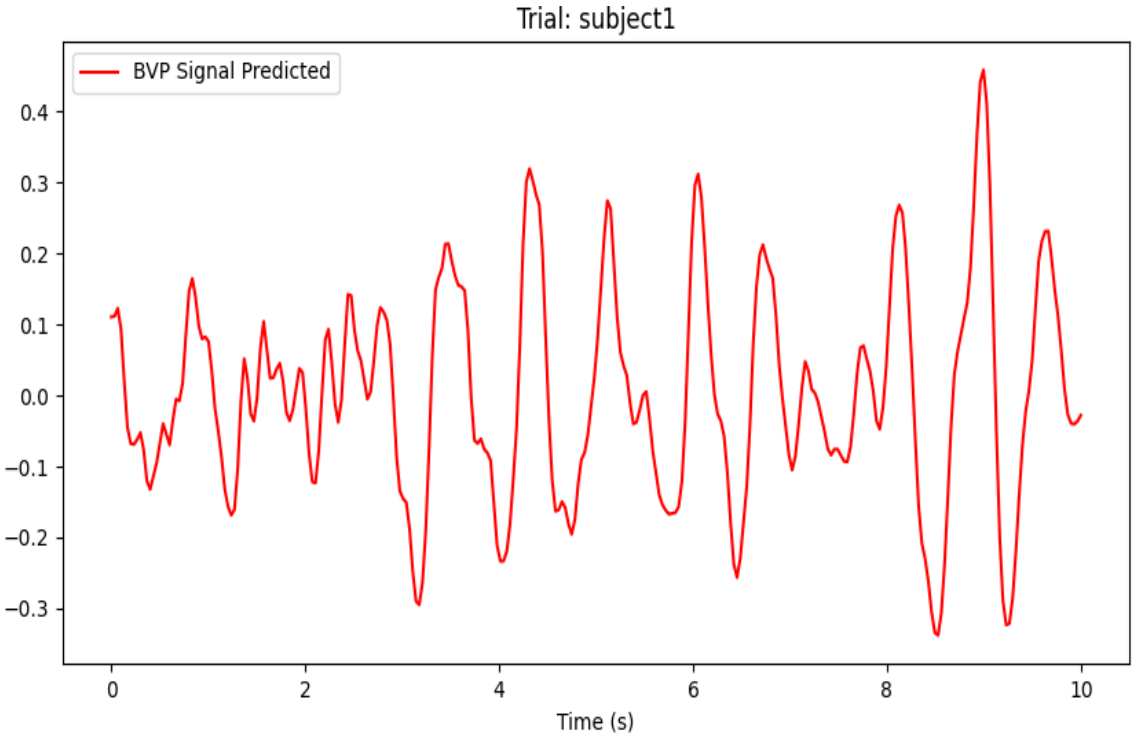
7. **DeepPhys Model to obtain BVP:** Blood Volume Pulse (BVP) is a measure of the change in blood volume within the microvascular bed of tissue, traditionally calculated by analyzing the light absorbed or reflected by the skin. This process involves shining light on the skin, where the contraction and relaxation of veins modulate the amount of light absorbed, which is then

captured by sensors to determine the pulse.

The **DeepPhys** model is a convolutional neural network (CNN) designed for video-based physiological measurement, specifically to estimate BVP from video inputs. The model takes two inputs: the difference between consecutive video frames (motion input) and the raw video frames (appearance input), and processes them through two parallel branches.
The motion branch extracts motion-related features using a series of convolutional layers, while the appearance branch extracts features related to the overall image content.

Attention mechanisms are applied in the appearance branch to focus on the most relevant regions of the image that contribute to BVP estimation. These attention maps are generated by apperance_att_conv1 and apperance_att_conv2, which are then normalized using custom Attention_mask layers. These attention maps modulate the features extracted by the motion branch, effectively "gating" the motion features to emphasize the areas that are important for BVP estimation.

The features from both branches are pooled, combined, and passed through a series of fully connected layers to produce a final BVP prediction. Dropout layers are used throughout to prevent overfitting by randomly setting a portion of the input features to zero during training. The use of both motion and appearance features, combined with attention mechanisms, allows the model to robustly estimate BVP even in challenging conditions, where traditional methods might fail.



8. **Heart Rate Calculation:** The heart rate in beats per minute (BPM) is calculated by analysing a Blood Volume Pulse (BVP) signal, which typically resembles a periodic waveform with peaks and troughs, similar to a sine wave. The BVP signal reflects the rhythmic changes in blood volume corresponding to heartbeats. The code identifies the peaks in this waveform, each peak representing a heartbeat, while filtering out noise by focusing on peaks that meet specific criteria in height and spacing. The time intervals between consecutive peaks, known as inter-beat intervals (IBIs), are calculated to determine the average time between heartbeats. This average IBI is then used to calculate the heart rate in BPM by considering the signal's sampling rate, assumed to be 30 samples per second. The estimated heart rate is finally output. We have used **Biosppy** library to convert the raw bvp signal into heart rate. This was done as the library is specifically designed to process ppg signals. The results of the library are more consistent.

9. **Respiration Rate Calculation:** The respiration rate from a Blood Volume Pulse (BVP) signal is calculated by first isolating the respiration-related frequencies within the signal, typically ranging from 0.1 to 0.5 Hz. This is done using a band-pass filter to retain only the components associated with breathing. The filtered signal's envelope, representing the amplitude variations, is then calculated, and peaks are detected within this envelope, corresponding to the breathing cycles. A binary signal is created where detected peaks are marked, indicating the timing of each breath. This binary signal is then analyzed to find the intervals between consecutive breaths by identifying where the signal transitions from zero to one, which marks the onset of a breath. These intervals are converted into respiration rate in breaths per minute. The code includes error handling to provide a default respiration rate if an issue arises during the calculation.

10. **Oxygen Saturation Level Calculation:** The estimation of SpO2 (oxygen saturation) is done by analyzing the mean and standard deviation of the blue and red color channels from a series of video frames focused on face regions. Each frame is processed to extract the blue and red channels, from which the mean and standard deviation values are computed. These values are then stored across all frames, with specific attention given to the blue and red channels. The SpO2 is calculated using a formula that involves the ratio of the standard deviation to the mean of the red and blue channels, scaled by constants A and B, and averaged over all frames. The result provides an estimate of oxygen saturation from the video data.

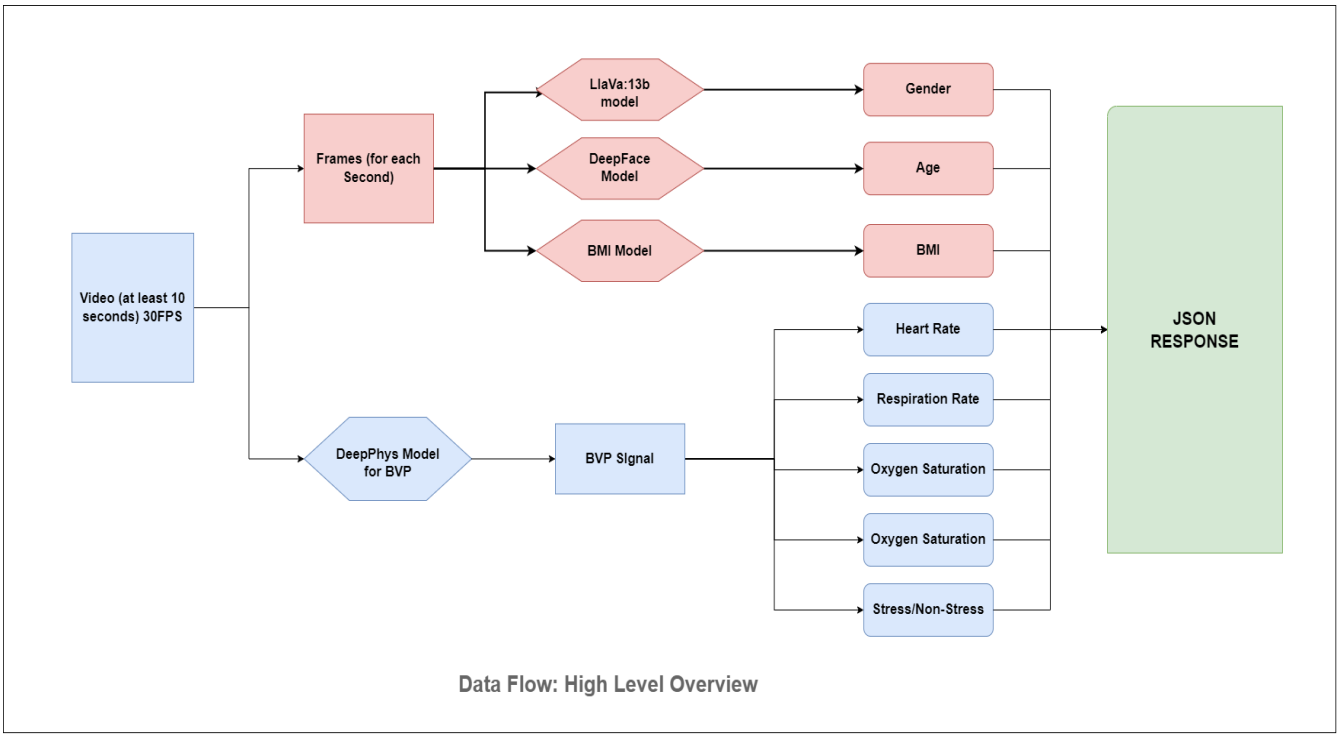
    The green channel is not used for estimating SpO2 because the key indicators for oxygen saturation are derived from the red and blue channels. The SpO2 calculation relies on the relationship between the standard deviation and mean of these two channels, which are more directly related to the physiological signals being analyzed. The red channel is crucial as it corresponds to the wavelengths of light that are most sensitive to changes in blood oxygen levels, making it particularly useful for detecting variations in blood volume and oxygen saturation. The blue channel, while not directly related to oxygen saturation, provides complementary information that helps refine the calculation by normalizing the signal. In contrast, the green channel does not contribute significantly to this specific calculation and is therefore omitted to simplify the process and focus on the channels with the most relevant information for SpO2 estimation.

11. **Stress/ non-stress:** The Heart Rate Variability (HRV) features are used to assess stress levels based on time-domain and frequency-domain metrics. Time-domain features are statistical measures derived from the RR intervals (the time between successive peaks in the PPG signal). **SDNN (Standard Deviation of NN Intervals**) represents the overall variability in heart rate and reflects how the interval between heartbeats varies over time. Higher SDNN values generally indicate a more relaxed state, while lower values suggest stress. Frequency-Domain

Features are derived from the power spectral density of the RR intervals. **LF/HF Ratio (Low Frequency / High Frequency Ratio)** compares the power in the low-frequency band to the power in the high-frequency band of the heart rate variability spectrum. A higher LF/HF ratio is often associated with stress and sympathetic nervous system dominance, while a lower ratio is associated with relaxation and parasympathetic dominance. If the SDNN value is lower than the threshold (after adjusting for the scale) or if the LF/HF ratio is higher than the threshold (after adjusting for the scale), the function will indicate stress. This means:

- Low SDNN: If the variability in RR intervals is too low, it suggests less adaptability and potential stress.
- High LF/HF Ratio: If the balance between the low-frequency and high-frequency components is skewed towards low frequency, it indicates more sympathetic (stress-related) activity compared to parasympathetic activity.

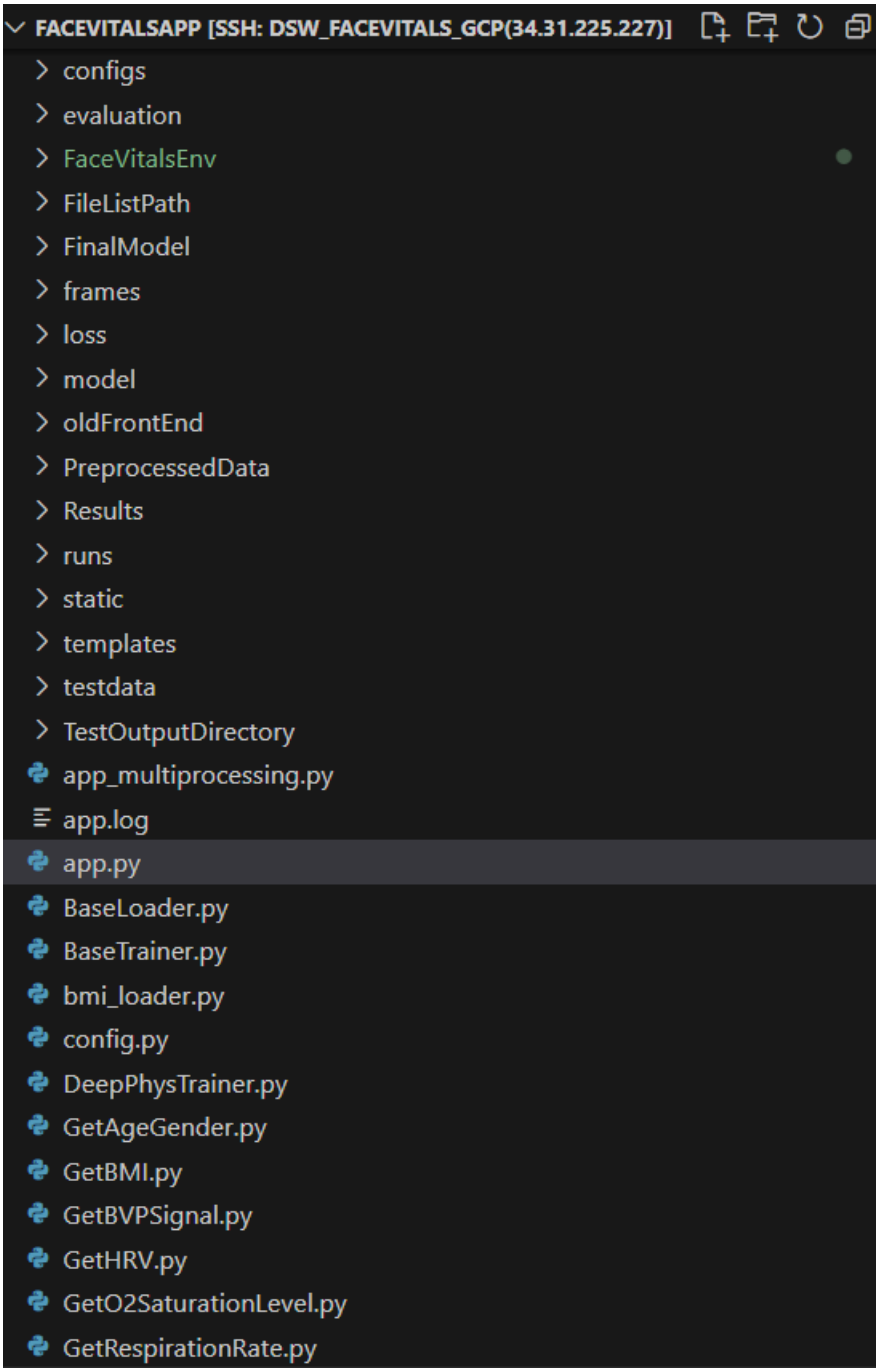
Data Flow



Data Flow: High Level Overview

Code Structure

- The Face Vitals code is developed on Flask framework.



- The config folder consists of configuration settings for DeepPhys model used to predict BVP signal from webcam videos. It has configurations like the oath of the model, output path to save pickle file containing BVP signal, chunk size in number of frames, and so on.
- FileListPath folder contains a .CSV file containing the associated numpy file for the test subject or the current user. It will be more beneficial when concurrent hits are done to the application, by maintaining segregation between different webcam videos and associated numpy files.
- FinalModel folder contains the pickle or saved model file.

- Frames folder consists of the frames that are used for age, gender and BMI prediction.
- Loss folder consists of python scripts that calculate the loss metric for DeepPhys model.
- Model folder consists of python scripts required to build a DeepPhys model.
- PreprocessedData folder contains the interim results in .numpy format.
- Results folder contains the final results like the .pickle file containing the BVP signal.
- Runs folder contains the output of current run. However, it is compatible in Linux OS. In Windows, since the name of run is saved using the name of all the transformations, it results in error.
- TestOutputDirectory folder also contains the output of current run which is currently ebing used and its path is set in config as well as GetBVPSignal.py file.
- Testdata folder contains a folder called subject1 which contains the current video being processed. The directory structure is maintained to replicate the UBFC-rPPG dataset.
- Config.py is a configuration python script that updates the configuration dynamically.
- The app.py is the main file that executes the Face Vitals application. It contains the homepage and the function to handle the POST request to gather all vitals and send response back to the client.
- The app_multiprocessing.py file is similar to app.py. It incorporates the use of multiprocessing module in python to carry out the task sequentially. Currently, we are not using it due to limited infrastructure. Once the GPU is increased to 20GB or above it can be used for parallel execution.
- BaseLoader and BaseTrainer are the base classes for the dataloaded and DeepPhysTrainer. They are used to retrain the DeepPhys model.
- The python scripts starting from Get like, GetBVPSignal, GetAgeGender are used to predict BVP signal, age and gender and so on respectlively.
- Predictions for gender is currently disable for deep face instead of that we are using LLava llm model for prediction. Functions to use use llava and predict output are added in app.py
- The app.log file contains the logs of the runs of the application like preprocessing steps and the intermediate results.
- In order to run the Face Vitals app, the app.py is the file to execute.

Setup

Prerequisites

- A GPU enabled system for training DeepPhys and BMI model. It can be done on CPU but will be more time intensive and slow.
- A system that has webcam with a resolution of atleast 640 X 480.
- A Linux server or cloud to host the application.
- RAM of 16 GB or above.
- GPU 16 GB or above.
- Python version 3.10 is required.
- Tensorflow==2.15.0. There is a dependency on this version with DeepFace library that allows us to call Age and Gender client rather than using the high level analyze function that takes more time in execution.

DeepPhys Configuration

- The config folder contains the configuration file for the DeepPhys model in the Face Vitals app. The cofig.py is a python script that dynamically updates the config for the DeepPhys model.

- Since heart rate, respiration rate, oxygen saturation level, and stress are calculated from BVP signal, the configuration of this model should be handled gracefully, to handle concurrent requests and proper configuration.
- Some basic configurations which can be updated are:
  Note: The configurations with comments are the one that are updated by the developer. Since we are extracting face vitals each time with new request, the DO_PREPROCESS is always set to True and should not be changed to False.

```
BASE: ['']
TOOLBOX_MODE: "only_test"      # "train_and_test"  or "only_test"
TEST:
  METRICS: ['MAE', 'RMSE', 'MAPE', 'Pearson', 'SNR', 'BA']
  USE_LAST_EPOCH: True
  OUTPUT_SAVE_DIR: "./Results/UBFC-rPPG_DeepPhys_UBFC-rPPG_outputs.pickle"
  DATA:
    FS: 30
    DATASET: UBFC-rPPG
    DO_PREPROCESS: True                 # if first time, should be true
    DATA_FORMAT: NDCHW
    DATA_PATH: "./testdata"       # need to be updated
    CACHED_PATH: "./PreprocessedData"    # Processed dataset save path, need to be updated
DEVICE: cuda:0
NUM_OF_GPU_TRAIN: 1
LOG:
  PATH: runs/exp
MODEL:
  DROP_RATE: 0.2
  NAME: DeepPhys
INFERENCE:
  BATCH_SIZE: 8
  EVALUATION_METHOD: FFT        # "FFT" or "peak detection"
  EVALUATION_WINDOW:
    USE_SMALLER_WINDOW: False        # Change this if you'd like an evaluation window smaller than the test vide
    WINDOW_SIZE: 10        # In seconds
  MODEL_PATH: "./FinalModel/UBFC-rPPG_DeepPhys.pth"
```

The path to the saved model is provided in the MODEL_PATH.

Ollama and LLava Model Setup for gender prediction

- **Install ollama for linux:**
  curl –fsSL https://ollama.com/install.sh | sh

- **Ensure ollama is installed by running:**
  ollama –help

- **Download the LLaVA:13b model:**
  ollama pull llava:13b
  ollama list (To verify whether the model is installed)

- **Serve the LLaVA Model on Localhost:**
  ollama serve llava:13b --host 127.0.0.1 --port 11434
  ollama serve llava:13b --host 0.0.0.0 --port 11434 (to serve on public IP[optional])

  use this API url to access llava in application : “http://localhost:11434/api/generate” (use public IP in case of localhost for remote access)

Project Setup for linux

1. **Create the Projects Directory**:
   mkdir Projects

2. **Navigate into the Projects Directory:**
   cd Projects

3. **Create a Virtual Environment using venv:**
   python -m venv Facevitals_venv (You can use Conda or virtualenv; just ensure that the environment is created with Python 3.10)

4. **Activate the Virtual Environment:**
   source Facevitals_venv /bin/activate

5. **Copy and Paste the application files into the Projects directory.**

6. **Install the dependencies using pip:**

   pip install -r requirements.txt (make sure that tensorflow 2.15.0 is install or else reinstall it manually and for pytorch find from https://pytorch.org/get-started/locally/ that is compatible with your gpu cuda and os version)

7. **Run application:**
   Python app.py

Testing

- **Age and Gender:** The age and gender are tested manually as they are itself known to the user. The gender model is required to predict accurately. The age model is allowed to have an error margin of 2 years.
- **BMI:** The Body Mass Index can be calculated online suing weight and height of the user which can be compared against the predictions of BMI model.
- **Heart Rate:** The heart rate is calculated using an Apple watch or any other fitness device that monitors it. An error rate of 5-10 BPM is acceptable but not desirable.
- **Respiration Rate:** The respiration rate is calculated using an Apple watch or any other fitness device that monitors it. The normal range for respiration is 12 – 20 breaths per minute at resting condition.
- **SpO2 Level:** The oxygen saturation level is calculated using oximeter or any fitness device that monitors it. SpO2 level is normally above 95%.

Deployment (On Linux)

- **General command to keep Linux updated**
  sudo apt update
  sudo apt upgrade

- **Install NGINX and configure nginx.**
  sudo apt install nginx
  sudo systemctl start nginx
  sudo systemctl enable nginx
  sudo systemctl status nginx

- sudo ufw allow 'Nginx HTTP'
  sudo ufw allow 'Nginx HTTPS'

- sudo systemctl reload nginx

- **Configure Nginx**: Create an Nginx server block configuration file in the sites-available folder:

  sudo nano /etc/nginx/sites-available/FaceVitalsApp

```
server {
    listen 80;
    server_name 34.31.225.227;

    # Redirect HTTP to HTTPS
    return 301 https://$host$request_uri;
}

# This block is to implement HTTPs request with SSL certifcates
server {
    listen 443 ssl;
    server_name 34.31.225.227;

    # These certificates are provided by DSW
    ssl_certificate /etc/nginx/datasciencewizards.ai.crt;
    ssl_certificate_key /etc/nginx/datasciencewizards.key;

    # Security settings
    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_ciphers
'TLS_AES_128_GCM_SHA256:TLS_AES_256_GCM_SHA384:TLS_CHACHA20_POLY1305_SHA2
56:RSA+AES:!aNULL:!eNULL:!MD5:!PSK';
    ssl_prefer_server_ciphers on;

    # Additional settings
    ssl_session_cache shared:SSL:10m;
    ssl_session_timeout 10m;
```

```
    ssl_session_tickets off;

    # Set max body size to 50MB that is it can accept 50 MB of data
    client_max_body_size 50M;

    # Timeout settings
    client_body_timeout 300s;
    client_header_timeout 300s;
    send_timeout 300s;
    proxy_read_timeout 300s;
    proxy_connect_timeout 300s;
    proxy_send_timeout 300s;

    # Keep-alive settings
    keepalive_timeout 65s;
    proxy_http_version 1.1;
    proxy_set_header Connection "";

    include /etc/nginx/mime.types;

    # The request is forwarded to http://127.0.0.1:8000 where the application is running
locally in the linux server or VM
    location / {
        proxy_pass http://127.0.0.1:8000;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }
}
```

 # Create symbolic link of this app in sites-enabled folder

```
sudo ln -s /etc/nginx/sites-available/CattleMuzzleApp /etc/nginx/sites-enabled
```

- **Create a service file**

```
cd /etc/systemd/system
```

 # you can keep any name of the service. Ensure extension is .service

```
sudo nano FaceVitalsApp.service
```

 # content of the file created above

```
[Unit]
Description=Gunicorn instance to serve your Face Vitals application
After=network.target

[Service]
User=gcpuser
Group=www-data
```

```
WorkingDirectory=/home/gcpuser/Projects/FaceVitalsApp
Environment="PATH=/home/gcpuser/Projects/FaceVitalsApp/FaceVitalsEnv/bin"
ExecStart=/home/gcpuser/Projects/FaceVitalsApp/FaceVitalsEnv/bin/gunicorn --workers 4 --bind 127.0.0.1:8000 --timeout 600 app:app

[Install]
WantedBy=multi-user.target
```

- **Follow the below steps to enable ,disable & check the status of your service**
  sudo systemctl daemon-reload **#to include new changes in the service file**
  sudo systemctl start FaceVitalsApp.service
  sudo systemctl stop FaceVitalsApp.service
  sudo systemctl reload FaceVitalsApp.service

Maintenance and Troubleshooting

- **Logging**
  The application consists of app.log file. This is the first file to check what operations have been done in the request. It also includes the error description. In order to troubleshoot add print statement and use exception handling effectively and log every operation.
- **Payload**
  The payload is the data that is sent from client or front end to the flask server. Ensure that the request from front end is send correctly and the payload is received as required.
- **Response**
  The flask server is expected to return a response for each request even if it fails or there is a runtime error. If a response is not returned it should be handled using exception handling and sending the reason of failure as response. Currently we are working on returning response if the CUDA memory is utilized completely.