Alora: Multimodal Stress Detection Chatbot Using NLP And CV

Darshan Jain, Veer Raje, Akshad Jain, Dnyanesh Sawant, Pratik Kanani

Dept. of Artificial Intelligence and Data Science

Dwarkadas J. Sanghvi College of Engineering, Mumbai, India

{jaindarshan1005, raje.veer95, akshadjain25, sawantdnyanesh511,pratikkanani123}@gmail.com

***Abstract*—Early acute stress detection is a key challenge in the treatment of mental illness. Current detection mechanisms are generally inadequate for effective, real-time handling of the user stress levels. The proposed paper presents a new stress detection method utilizing a conversational chatbot that combines Computer Vision (CV) and Natural Language Processing (NLP). The system identifies the user inputs to the text through an analysis based on an NLP-based Support Vector Classifier (SVC) for stress linguistic features with 85.47% validation accuracy and performance improvement over a Random Forest baseline model. Concurrently, the CV module utilizes a trained Convolutional Neural Network (CNN) model based on the PURE dataset for predicting real-time heart rate based on remote photoplethysmography (rPPG) to detect a heart rate. Outputs from the NLP and CV models are combined via a lightweight SVC ML model, classifying stress into low, medium, and high. The chatbot then offers tailored feedback and mental well-being advice depending on the anticipated stress level. Experimental results indicate a considerable improvement in performance by the multimodal approach compared to unimodal systems. The model achieved an impressive accuracy of 90.5% in classifying the level of stress, and the system's minimum response time of less than 2.5 seconds per interaction adds feasibility for real-time deployment. This work demonstrates that the combination of textual and physiological signals offers superior diagnostic reliability, providing a basis for accessible and non-intrusive psychological assessment tools.**

***Keywords*— r-ppg, cv, nlp, mental health, heart rate**

I. Introduction

Psychological stress, compounded by increasing socio-economic loads, school competition, and global health pandemics, has become a humongous public health issue among working adults and youth[1]. Conventional stress detection technologies that primarily depend on clinician-rated scales, self-reporting questionnaires, or even certain wearable sensors have considerable shortcomings [2],[3]. The technologies are intrusive, time-consuming, expensive, necessitating trained professionals or equipment, and might not be deployable to the huge population as conveniently, particularly in resource-poor environments such as developing economies[1]. These challenges lead to delays in diagnosis and treatment, and hence lead to severe physical and mental health outcomes.

The recent developments in Artificial Intelligence (AI) have led to the development of automatic systems to detect states of stress through non-invasive modalities[4],[5]. In particular, Natural Language Processing (NLP) can process stress signals in the form of text-based inputs[6],[7], whereas Computer Vision (CV) provides methods to track physiological signals[8],[9]. For example, remote photoplethysmography (rPPG) provides methods to estimate heart rate from video, a well-established physiological signal of stress[8],[10-16]. Although AI-based health agents have promising research, synergistic integration of NLP (for text-based stress signals) and CV (for rPPG-based physiological stress signals) in a chatbot platform for comprehensive, real-time stress detection is an untapped field.

Current AI-based stress assessment systems have become popular in healthcare due to their ability to mimic human conversation; however, most current systems lack advanced stress assessment functionality and often run unimodally, analyzing either text or physiological data independently. This intrinsic constraint can limit their contextual awareness and accuracy, leading to higher false positives or misclassifications of stress states. There is a need for an integrated, real-time, and user-oriented approach that can effectively integrate linguistic and physiological cues to assess stress reliably[17]. This is particularly well-suited to the recovery of weak physiological stress markers, e.g., rPPG-detectable facial video-based heart rate fluctuations [8],[18],[19], combined with language based stress indicators[6].

In this paper, a new multimodal chatbot solution is presented to tackle these issues. The system takes advantage of sophisticated NLP methods for processing stress indicators as cues from the user's text input. Parallelly, the CV module applies a Convolutional Neural Network (CNN) model, which has been trained on datasets like the PURE dataset or others[29], to provide real-time estimation of heart rate based on remote photoplethysmography(rPPG)[11],[19], applied as a physiological cue for the analyzing stress. Machine Learning algorithms allow the system to detect stress patterns from multimodal combined information[20]. The overall goal is to create a user-friendly, non-invasive method that tracks and processes textual and rPPG-derived

physiological markers in real-time, detects the user's stress levels, and provides empathetic, personalized feedback and coping strategies. Robust data privacy practices are an integral part of the system design to establish user trust.

The major contributions of this paper are:

- A novel multimodal chatbot framework that integrates NLP for text stress analysis and CV for rPPG-based physiological stress indication (heart rate).
- An easy-to-use and accessible system for non-intrusive psychological stress screening and customized feedback based on aggregated textual and physiological evaluation.
- Combinating MSR/RGB and Temporal Wavelet methods for robust detection under various lighting conditions and subtle motion.

II. Related Works

Stress detection by AI has gained considerable momentum with growing mental problems and growing data availability[1],[21],[22]. Existing research work relies mainly on NLP for processing textual cues or CV to acquire physiological signals. This section summarizes existing research work and makes references to multimodal studies, highlighting areas where our system bridges the gap.

Stress Detection using NLP (Text): Textual analysis for psychological stress indicators is an emerging field of study. Various machine learning and deep learning models have been used to detect stress-indicating linguistic features. As demonstrated, the use of sentiment analysis from textual data in an interview transcript to classify PTSD individuals with high accuracy through the use of machine learning models[6]. These works illustrate the ability to leverage lexical and syntactic features derived from text as predictors of stress and related mental health illness[7]. Even text-based methods face certain challenges. Decision interpretation by models is hard, and massive sets of annotated data are usually needed to train potent models. In addition, self-expression using writing is not necessarily indicative of the real status of the user since people would willingly or unintentionally hide or downplay the degrees of their stress while writing[2],[3],[23],[24]. It is such a limitation that necessitates utilization of other or other alternative, or complementary modalities.

Physiological Stress Indicator Detection using CV (rPPG): CV techniques offer non-invasive means of estimating physiological stress correlates. Remote photoplethysmography (rPPG) is one of the prominent technologies for estimating heart rate (HR) and heart rate variability (HRV) from video, typically by sensing subtle color changes on the face using standard cameras[8],[10] ,[14-16],[25]. HR and HRV differences are well established physiological indices of the autonomic nervous system's response to stress[18]. Several studies have explored rPPG for stress assessment. We can use cameras to obtain transdermal optical imaging signals (a form of rPPG) and derive HR/HRV values indicating levels of stress, in a comparable way to ECG[18]. This paper highlighted a comprehensive approach with rPPG analysis and deep learning for better stress detection[8]. Deep learning techniques, particularly Convolutional Neural Networks (CNNs), have been employed with greater frequency for rPPG signal extraction and HR estimation to improve robustness[9],[11],[12],[13],[19],[20]. For instance, Jaiswal and Meenpal (2022) proposed rPPG-FuseNet for non-contact HR estimation through the fusion of RGB/MSR signals[11] and also explored HR estimation from spatiotemporal feature images[12]. Robust region of interest selection to improve rPPG accuracy[13][32]. Despite all of these advances, challenges such as sensitivity to motion artifacts, variations in environmental lighting, and overall low signal-to-noise ratio of the physiological signal still pervades[8],[26]. While deep learning models promise to mitigate such difficulties over more traditional signal processing paradigms[9],[19], robust rPPG-based stress estimation in unconstrained real-world applications remains a thriving research area. Furthermore, video-based physiological monitoring also poses privacy concerns that need to be resolved by careful system design. The use of smartphone cameras for such physiological measurements has also been highlighted[18].

Multimodal Stress Detection: Recognizing the limitations of unimodal systems, researchers are experimenting with combining more than one stream of data for more robust stress detection. Multimodal models that combine NLP textual features with other means of measurement are able to supply a better representation of the user's state[17]. Demonstrated the fusion of image and text data in human-robot interaction for children with ASD, indicating possibilities of mixing different types of data[17]. Saeed and Trajanovski (2017) revealed ways in which multi-task neural networks could leverage physiological signals to personalize stress detection, highlighting benefits of combination of information and personalization[20]. Systematic reviews by Alreshidi et al. (2024) on ML with psychophysiological data in aviation[4], and Yaacob et al. (2023) on AI for BCI-based mental fatigue detection[5] also indicate the increasing interest in utilizing multiple sources of data or sophisticated AI for mental state assessment. Other modalities such as wrist-based Electrodermal Activity (EDA)[27] or keyboard and mouse dynamics[28] have also been investigated, further highlighting the trend towards multimodal or multi-sensor approaches. Brown and Nelson (2021) even suggested a system named SoDA for stress detection and mitigation, leveraging several inputs[22].

The motivation for multimodal fusion is that complementary information sources may increase robustness and accuracy, compensating for noise or ambiguity in single modality. Nonetheless, proper fusion of diverse sources like text and physiological signals is not as easy. Challenges include synchronizing data streams recorded at varying rates and to temporally align features. Optimal fusion strategies (e.g., feature-level, decision-level, hybrid) that best balance the contribution of each modality is an ongoing research topic. Large, synchronized multimodal datasets specifically

capturing natural language interactions and related rPPG signals under diverse stress conditions are not available, thus challenging the design and rigorous testing of such integrated systems.

Synthesis and Research Gaps: Literature reviewed above offers significant breakthroughs in application of NLP over text-based stress analysis and CV (in the form of rPPG) towards stress physiology signaling. Multimodal modalities are intriguing but face the practical challenges of obtaining the data, as well as data fusion processes. Certain prominent gaps are persisting in place:

- Real-Time Integration of Chatbots: Majority of current research is dependent on component-level validation. Integrated systems incorporating both NLP and CV in a functional conversational agent (chatbot) for real-time assessment and feedback on stress do not exist.
- Generalizability and Accessibility: Majority of models are trained on specific datasets or laboratory environments. The establishment of solid, accessible systems deployable in daily hardware (webcams, smartphones) in broad populations remains unattainable.
- User Experience and Ethics: Incorporating physiological monitoring (rPPG) into a chatbot must also be considerate of user acceptability, privacy, consent, and ethical display of feed concerning mental state.

Our work aims to design and evaluate a multimodal chatbot with real-time NLP processing of user input combined with simultaneous CV-based rPPG estimation of heart rate through a fusion mechanism, yielding a seamless integrated stress assessment and personalized user feedback in an effortless manner.

III. Proposed Method

This paper details "Alora," a multimodal chatbot for stress detection. The system integrates NLP for textual analysis with CV for rPPG-based heart rate estimation from video. Outputs from both modalities are fused for a comprehensive stress assessment, as detailed in the following sections:

A. NLP and Classification Approach

B. Physiological Image Signal Based Stress Prediction (CV)

C. Multimodal Fusion and Final Stress Prediction

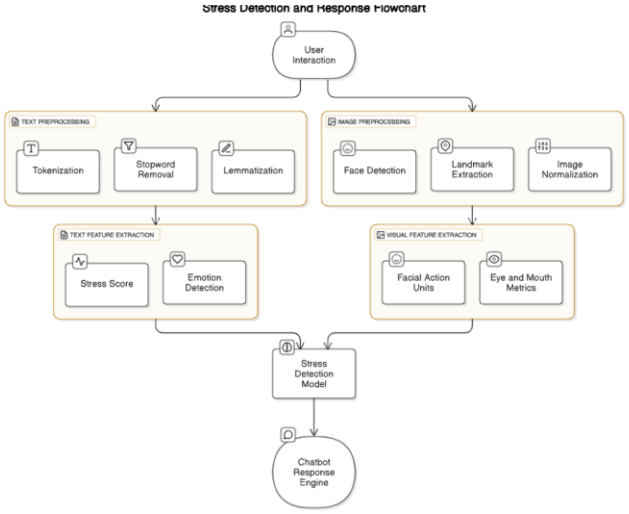


Fig 1. Architecture of Alora

A. NLP and Classification Approach:

1. Text Preprocessing and Feature Engineering:

- Representation: A Bag-of-Words (BoW) approach was followed to represent unstructured textual elements as structured numerical vectors.
- Tokenization & Quantification: Text was tokenized into unigrams, and their frequencies within documents were quantified.
- Vocabulary Capping: Vocabulary was capped at the 3,000 most common words to ensure computational efficiency and give implicit regularization against noisy/rare tokens. This was carried out with Scikit-learn's CountVectorizer.

2. Classification Model:

- Algorithm: A linear Support Vector Classifier (SVC) for binary classification was chosen as it is strong in high-dimensional text data
- Kernel: A linear kernel keeps interpretability high and computational complexity low.
- Objective: The model aimed to identify the optimal hyperplane for separating participants with and without self-reported anxiety.

3. Model Development and Evaluation:

- Data Splitting: Samples were allocated into training, testing, and validation subsets (three-phase split) for model fitting, initial performance estimation, and final real-world applicability assessment.
- Hyperparameter Tuning: The SVC's regularization parameter ‘c’ was systematically fine tuned by exploring values between 0.01 and 0.49 (increments of 0.01). For each ‘c’ value, the model was retrained, and performance was measured on all three datasets.
- Evaluation Metric: Model accuracy (proportion of correctly predicted labels) was the primary metric,

chosen due to balanced class distribution.

- Stability Assessment: Accuracy trends across the range of ‘c’ values were analyzed to assess model stability and reliability. A line plot visualized this relationship.

4. Implementation Details:

- The pipeline was implemented in Python using the Scikit-learn library for preprocessing, model building, and evaluation.
- A total of 49 models were trained, corresponding to the different ‘c’ values.
- This approach leveraged interpretable and efficient machine learning techniques for classifying anxiety from survey responses, demonstrating a pathway for scalable, automated mental health risk assessment based on textual data.

Model	Random Forest	Support Vector
Training Accuracy:	74.79%	91.59%
Validation Accuracy:	74.85%	85.47%
Test Accuracy:	75.87%	85.15%

Table.1 RF and SVC Performance Measurement

The SVC model performed much better than the counterpart. The Random Forest had moderate performance with stable accuracy but had a limited ability to learn complex relationships. However, the SVC produced a significantly higher training accuracy, reflecting a good fit to the training data, and had strong generalization with test and validation accuracies of approximately 85%.

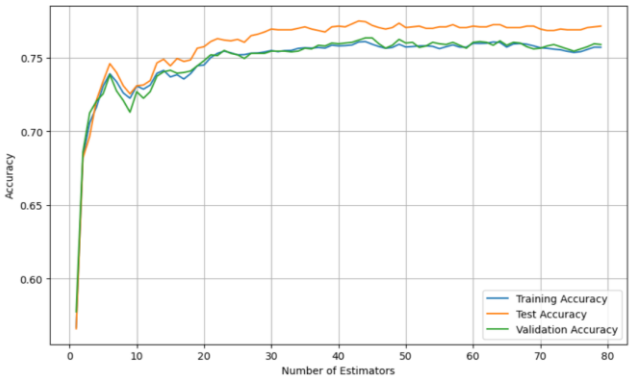


Fig 2. Accuracy vs No. of Estimators for Random Forest

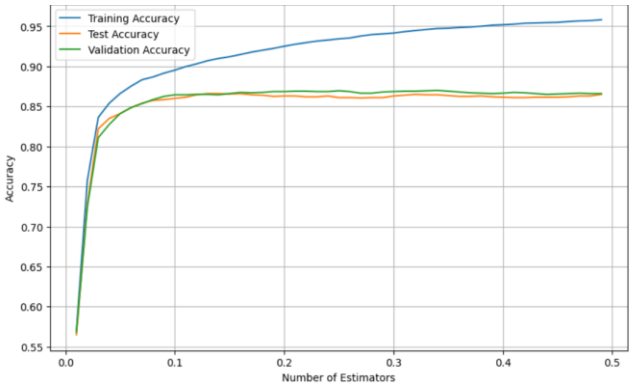


Fig 3. Accuracy vs No. of Estimators for SVC

```
Algorithm 1: Emotion Detection from Text
Input: User text input User_Text
Output: Predicted emotion Predicted_Emotion

Load the pre-trained Support Vector Classifier (SVC) model NLP_Model and the vectorizer Vectorizer from saved files.
Preprocess User_Text:
    Tokenize User_Text into tokens using the same tokenization method as in training.
    Pad the tokenized sequence to ensure it has the same length as the sequences used in training.
Vectorize the preprocessed text using Vectorizer.transform(preprocessed_text) → vectorized_text.
Predict the emotion using NLP_Model.predict(vectorized_text) → Predicted_Emotion.
Return Predicted_Emotion.
```

```
Algorithm 2: Stress Score Prediction with Sentiment Clustering
Input: Dataset file path Dataset_Path
Output: Trained stress prediction model Stress_Model, stress scores Stress_Scores

Initialization:
    Load the dataset from Dataset_Path → df.
    Load the SentenceTransformer model ('all-MiniLM-L6-v2') → Model.
    Set number of clusters to 4 (for joy, sadness, anger, fear) → Num_Clusters.
    Initialize K-Means clustering with Num_Clusters and random_state=42 → KMeans.
Text Embedding:
    Extract text content from df['content'] → Texts.
    Use Model to encode Texts into embeddings → Embeddings.
    Normalize Embeddings to unit length for cosine similarity → Embeddings.
Sentiment Clustering:
    Apply KMeans to Embeddings to get cluster labels → Cluster_Labels.
    Create a mapping of clusters to dominant sentiments:
        For each cluster in range(Num_Clusters):
            Identify the most frequent sentiment in the cluster → Dominant_Sentiment.
            Map cluster to Dominant_Sentiment → Cluster_to_Sentiment.
        Ensure all sentiments (joy, sadness, anger, fear) are represented:
            Track used sentiments → Used_Sentiments.
            For each cluster, if its Dominant_Sentiment is already used, assign an unused sentiment → Cluster_to_Sentiment.
Stress Score Calculation
```

Identify indices of 'joy' sentiment in df → Joy_Indices.

Extract embeddings for 'joy' sentiment → Joy_Embeddings.

Compute the centroid of Joy_Embeddings → Joy_Centroid.

Calculate cosine similarity of each embedding to Joy_Centroid → Cosine_Similarities.

Compute initial stress scores as (1 - Cosine_Similarities) → Stress_Scores (range [0, 2]).

Normalize Stress_Scores to [0, 1] using min-max normalization → Stress_Scores.

Scale Stress_Scores to [0, 5] by multiplying by 5 → Stress_Scores.

Assign Stress_Scores to df['stress_score'].

Model Training

Set features as Embeddings → X.

Set target as Stress_Scores → y.

Split X and y into training (80%) and testing (20%) sets with random_state=42 → X_train, X_test, y_train, y_test.

Initialize RandomForestRegressor with n_estimators=100 and random_state=42 → Regressor.

Train Regressor on X_train and y_train → Stress_Model.

Model Evaluation

Predict stress scores on X_test → y_pred.

Compute Mean Squared Error (MSE) between y_test and y_pred → MSE.

Compute Mean Absolute Error (MAE) between y_test and y_pred → MAE.

Compute R^2 Score between y_test and y_pred → R2_Score.

Print MSE, MAE, and R2_Score.

Model Saving

Save Stress_Model to "stress_regressor.pkl" → Stress_Model_File.

Return Stress_Model, Stress_Scores.

```

```

Algorithm 3: API Workflow for Multimodal Chatbot

Input: User text input User_Text

Output: Chatbot response Bot_Response, predicted emotion Predicted_Emotion, stress level Stress_Level

Initialization

Load the Gemini API key → api_key.

Establish a chat session with the Gemini LLM using api_key → Chat_Session.

User Input

Receive User_Text from the user via the chatbot interface.

Emotion Prediction

Call the emotion prediction API with User_Text → Predicted_Emotion.

Stress Score Prediction

Call the stress prediction API with User_Text → Stress_Score.

Map Stress_Score to Stress_Level (e.g., scale to 1-5).

Chatbot Response Generation

Construct a prompt including User_Text, Predicted_Emotion, Stress_Level, and the chatbot's persona/instructions → prompt.

Send prompt to Chat_Session → Bot_Response.

Return Bot_Response, Predicted_Emotion, Stress_Level.

```

The Stress Score Prediction module, as outlined in Algorithm 2, was specifically designed and extensively tested using the ISEAR (International Survey on Emotion Antecedents and Reactions) dataset [31]. This dataset comprises 7,666 text samples, each annotated with one of four primary emotions: anger, fear, joy, or sadness. For text embedding, the `all-MiniLM-L6-v2` SentenceTransformer model was used with vectors having 384 dimensions. The K-Means clustering process, as outlined, efficiently segmented the texts into these emotional clusters, followed by a corrective reassignment such that each of the four sentiments had a unique cluster.


Fig.4 K-Means Clustering on ISEAR

The RandomForestRegressor, with an initialization of 100 decision trees and training over an 80/20 split of the ISEAR data, was utilized to predict the continuous stress scores (0 through 5) obtained from the text embeddings. The regression model was good at the test set, with a Mean Squared Error (MSE) of 0.1689, a Mean Absolute Error (MAE) of 0.3239, and an R-squared ($R^2$) score of 0.7546. The $R^2$ score indicates that the model explains around 75.46% of the variance of the stress scores, thus legitimizing the validity of utilizing text embeddings for accurate stress level estimation.

### B. Physiological Image Signal Based Stress Prediction:

**1. Preprocessing Pipeline:** Input videos (e.g., MP4, AVI) were standardized to a consistent frame rate (30 fps) using OpenCV. The MediaPipe FaceLandmarker detected 478 facial landmarks per frame in video mode, providing normalized coordinates and timestamps. Three Regions of Interest (ROIs)—forehead, left cheek, right cheek—were defined using specific landmark indices. Raw pixel data for each ROI was extracted based on bounding boxes derived from denormalized landmark coordinates. Extracted ROIs were resized to a fixed ROI size (64x64 pixels) using area interpolation and converted to float32. Frames with failed detections or invalid ROIs were handled appropriately.

```

Algorithm 4: Video Preprocessing and ROI Extraction

Input : Input video path `input_path`, ROI landmark indices (Forehead, Cheeks).

Output : Sequence of standardized ROIs (Forehead, Left Cheek, Right Cheek) for each valid frame `roi_sequences`.

Convert `input_path` to 30 FPS video `converted_video_path` (Preserves all frames, sets fixed FPS).

Initialize `face_landmarker` (MediaPipe).

Initialize empty lists `forehead_rois`, `left_cheek_rois`, `right_cheek_rois`.

Open video capture for `converted_video_path`.

For each `frame` at `timestamp_ms`:

Detect face landmarks using `face_landmarker.detect_for_video(frame, timestamp_ms)` -> `detection_result`.

If `detection_result` contains landmarks:

Extract landmark coordinates `landmarks_coords` (Pixel values).

Define bounding boxes `bbox_forehead`, `bbox_left_cheek`, `bbox_right_cheek` using predefined indices on `landmarks_coords`.

For each `bbox`:

If `bbox` is valid:

Extract `roi_raw = frame[bbox]` (Slicing using bbox coordinates).

Clip `roi_raw` coordinates to frame boundaries (Ensure x1,y1 >= 0 and x2 <= W, y2 <= H).

If clipped ROI has area > 0:

Resize `roi_raw` to standard `ROI_SIZE` (e.g., 64x64) using `cv2.resize` -> `roi_standardized`.

Append `roi_standardized` to the corresponding list (`forehead_rois`, etc.).

Else: Append `None` (indicates failed extraction for this frame).

Else: Append `None`.

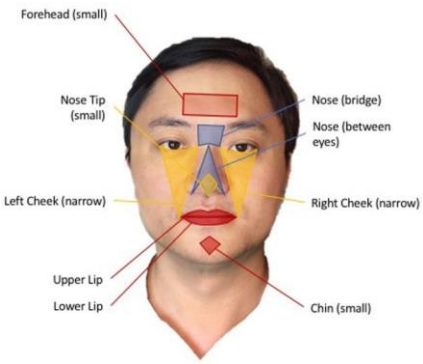
Else: // No landmarks detected

Append `None` to all ROI lists for this frame (maintains sequence alignment).

Release video capture (`cap.release()`) and `face_landmarker` (`face_landmarker.close()`).

Return `roi_sequences = { 'forehead': forehead_rois, 'left_cheek': left_cheek_rois, 'right_cheek': right_cheek_rois }`.

```




```

FOREHEAD_POINTS = [103, 67, 109, 10, 338, 297, 332, 333, 334, 296, 336, 285, 417, 351, 419, 197, 196, 122, 193, 55, 107, 66, 105, 104, 103]

LEFT_CHEEK_POINTS = [345, 340, 346, 347, 348, 349, 329, 371, 266, 425, 411, 352, 345] # Viewer's right

RIGHT_CHEEK_POINTS = [116, 111, 117, 118, 119, 120, 100, 142, 36, 205, 187, 123, 116] # Viewer's left

```

~ for MediaPipe

Fig.5 Best Region of Interests [18]

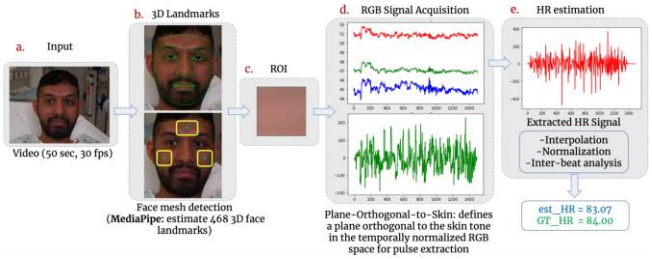



Fig.6 Best Region of Interests utilizing MediaPipe[13]

**2. Dual-Path Feature Extraction:** A dual-path strategy is used to extract complementary features from each standardized ROI:

2.1 Path A (Temporal Wavelet Features): Emphasized temporal dynamics. Frame differencing (diff_k(t) = current_roi_k(t) - prev_roi_k) was carried out. A 3-level 2D Discrete Wavelet Transform (DWT, 'db4' wavelet) was applied on each color channel to diff_k(t). Horizontal (HL) and Vertical (LH) detail subband projections were computed for each level. These projections were concatenated across levels and channels to create the wavelet feature vector wavelet_fv_k(t).

```

Algorithm 5: Path A Feature Extraction (Wavelet-Difference)

Input : Sequence of standardized ROIs `roi_sequence` (List of HxWxC frames or None), wavelet name `wavelet`, decomposition levels `levels`.

Output : Sequence of 1D Wavelet Feature vectors `wavelet_feature_sequence`.

Initialize `wavelet_feature_sequence = []`.

Initialize `prev_roi = None`.

For `current_roi` in `roi_sequence`:

If `current_roi` is not None and `prev_roi` is not None:

Calculate difference frame `diff_frame = current_roi.astype(np.float32) - prev_roi.astype(np.float32)`.

Initialize `combined_channel_features = []`.

For each channel `c` in `diff_frame`: // Iterate through C dimension

Perform multi-level 2D wavelet decomposition `pywt.wavedec2(diff_frame[:,:,c], wavelet, level=levels)` -> `coeffs`.

Extract Horizontal Detail (cH/HL) and Vertical Detail (cV/LH) subbands for each level `l` from 1 to `levels` from `coeffs`.

Compute Vertical Projection `Proj_HL_l = np.sum(HL_l, axis=0)` (Sum columns).

Compute Horizontal Projection `Proj_LH_l = np.sum(LH_l, axis=1)` (Sum rows).

Concatenate projections for all levels: `channel_feature_vector = np.concatenate([Proj_HL_1, Proj_LH_1, ..., Proj_HL_levels, Proj_LH_levels])`.

Append `channel_feature_vector` to `combined_channel_features`.

Concatenate all channel feature vectors -> `frame_feature_vector = np.concatenate(combined_channel_features)`.

Append `frame_feature_vector` to `wavelet_feature_sequence`.

Else: // Missing frame or first frame

Append `None` to `wavelet_feature_sequence`. // Maintain sequence alignment

Update `prev_roi = current_roi`. // Store current ROI for next difference calculation

Return `wavelet_feature_sequence`.

```

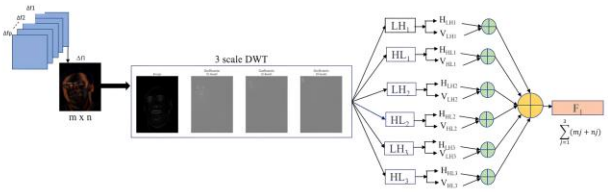



Fig. 7 Block diagram STI generation (HL and LH shows subbands of wavelet and ⊕sign shows the vector concatenation) [12]

2.2 Path B (Spatial Illumination-Robust Block Features): Emphasizing illumination robustness. Multi-Scale Retinex (MSR) was used on current_roi_k(t). The RGB ROI and MSR ROI were fused with saliency-weighted detail layers. The fused image (fused_k(t)) was segmented into N x N blocks, and mean color values by block were computed and flattened into the block feature vector current_block_fv_k(t).

```

Algorithm 6: Path B Feature Extraction (Illumination-Robust)

Input : Sequence of standardized ROIs `roi_sequence` (List of HxWxC frames or None), MSR scales `scales`, fusion kernel size `ksize`, number of blocks `num_blocks_sqrt`.

Output : Sequence of 1D Block Feature vectors `block_feature_sequence`.

Initialize `block_feature_sequence = []`.

For `current_roi` in `roi_sequence`:

If `current_roi` is not None:

// --- MSR Calculation ---

Initialize `msr_output` as zeros like `current_roi`.

Convert `current_roi` to float32 `roi_float`.

`weight = 1.0 / len(scales)`.

For each channel `c` in `roi_float`:

`log_channel = np.log10(roi_float[:,:,c] + epsilon)`.

For `scale` in `scales`:

Apply Gaussian blur `cv2.GaussianBlur(roi_float[:,:,c], (0,0), sigmaX=scale)` -> `blurred`.

`log_blurred = np.log10(blurred + epsilon)`.

`diff = log_channel - log_blurred`.

Add `weight * diff` to `msr_output[:,:,c]`.

`msr_roi = msr_output`.

// --- RGB-MSR Fusion ---

Compute base layers `base1 = cv2.blur(roi_float, (ksize, ksize))`, `base2 = cv2.blur(msr_roi, (ksize, ksize))`.

`fused_base = (base1 + base2) / 2.0`.

Compute detail layers `detail1 = roi_float - base1`, `detail2 = msr_roi - base2`.

Calculate saliency `saliency1 = np.abs(detail1)`, `saliency2 = np.abs(detail2)`.

`total_saliency = saliency1 + saliency2 + epsilon`.

Calculate weights `weight1 = saliency1 / total_saliency`, `weight2 = saliency2 / total_saliency`.

`fused_detail = weight1 * detail1 + weight2 * detail2`.

`fused_roi = fused_base + fused_detail`.

// --- Block Feature Extraction ---

`H, W, C = fused_roi.shape`.

`k = num_blocks_sqrt * num_blocks_sqrt`.

Initialize `block_features` array (k, C), dtype=float32.

Calculate block height `block_h = H // num_blocks_sqrt` and width `block_w = W // num_blocks_sqrt`.

`block_idx = 0`.

For row `r` from 0 to `num_blocks_sqrt` - 1:

For column `c_block` from 0 to `num_blocks_sqrt` - 1:

Extract `block` from `fused_roi` using `r, c_block` indices and `block_h, block_w`.

Calculate average values `avg_vals = np.mean(block, axis=(0, 1))` across channels.

Store `avg_vals` in `block_features[block_idx, :]`.

Increment `block_idx`.

Flatten `block_features` -> `frame_feature_vector = block_features.flatten()`.

Append `frame_feature_vector` to `block_feature_sequence`.

Else: // Missing frame

Append `None` to `block_feature_sequence`.

Return `block_feature_sequence`.

```

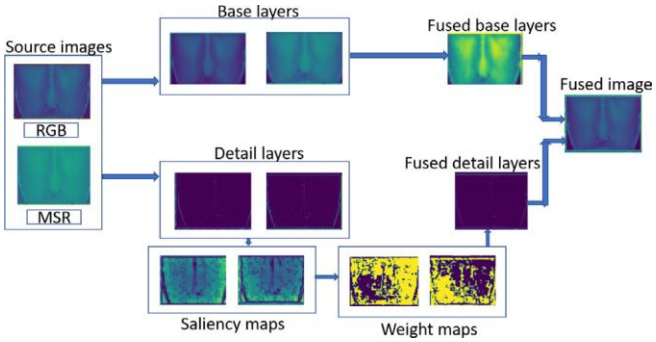



Fig.8 Block diagram of fusion algorithm [11]

Feature Combination: The temporal wavelet vector wavelet_fv_k(t) was concatenated with the spatial block vector of the previous frame prev_block_fv_k to create combined_fv_k(t) for every ROI, retaining both temporal change and prior spatial context.

**3. Spatio-Temporal Image (STI) Construction:** For each ROI, the sequence of combined_fv_k(t) vectors for a video clip was stacked vertically (np.vstack) and resized (to 3x224x224) with linear interpolation to build final_STI_k. The STIs of the three ROIs were stacked along the channel axis (np.stack) to build a single 3-channel input tensor (stacked_input_sti) for the CNN.

```

Algorithm 7: Spatio-Temporal Representation Generation

Input : Wavelet feature sequences `wavelet_features` (Dict: ROI -> List of vectors/None), Block feature sequences `block_features` (Dict: ROI -> List of vectors/None), target STI size `target_size`.

Output : Stacked Spatio-Temporal Image `stacked_sti` (3, H_target, W_target) or None.

Initialize `combined_features` (Dict: ROI -> List of combined vectors).

Determine sequence length `N` (from any non-empty feature list).

Initialize `prev_block_vectors` (Dict: ROI -> None).

For `t` from 0 to `N-1`: // Iterate through time steps

For each `ROI` (forehead, left_cheek, right_cheek):

`wavelet_fv_t = wavelet_features[ROI][t]` (Path A feature from difference t vs t-1).

`block_fv_t = block_features[ROI][t]` (Path B feature from frame t).

`prev_block_fv = prev_block_vectors[ROI]` (Path B feature from frame t-1, stored in prev iteration).

If `wavelet_fv_t` is not None and `prev_block_fv` is not None:

Concatenate `combined_fv = np.concatenate([wavelet_fv_t, prev_block_fv])`.

Append `combined_fv` to `combined_features[ROI]`.

Update `prev_block_vectors[ROI] = block_fv_t`. // Store current block features for next step's combination

Initialize `sti_dict` (Dict: ROI -> STI image).

For each `ROI`:

`feature_list = combined_features[ROI]`.

If `feature_list` is not empty:

Stack vectors vertically: `sti = np.vstack(feature_list)`.

Convert `sti` to float32.

Resize `sti` to `target_size` using `cv2.resize` with linear interpolation -> `resized_sti`.

`sti_dict[ROI] = resized_sti`.

Else: `sti_dict[ROI] = None`.

If all ROIs in `sti_dict` have a valid STI (are not None):

Stack STIs along the channel dimension: `stacked_sti = np.stack([sti_dict['forehead'], sti_dict['left_cheek'], sti_dict['right_cheek']], axis=0)`.

Return `stacked_sti`.

Else:

Return None.

```

**4. Heart Rate Estimation CNN:** ResNet18 was used as the backbone, pre-trained on ImageNet. The initial layer was modified to accommodate the 3-channel STI input. Standardization of inputs was included during the forward pass. Self-attention blocks were added following intermediate layers (e.g., layer2, layer4) to pay attention to salient features. Features were extracted in several stages, pooled by place through adaptive average pooling, and concatenated together for multi-scale feature fusion. A Multi-Layer Perceptron (MLP) regression head (with hidden layers, BatchNorm, ReLU, Dropout) projected the combined features to a single predicted HR value.
```

$$Average\ Pooling\ = \frac{1}{f\ *\ f} * \sum_{i=1}^{f} \sum_{j=1}^{f} x_{ij}$$

```
Algorithm 8: Enhanced HR-CNN Architecture & Forward Pass
Input  : Stacked STI `x` (Batch, 3, H, W).
Output : Predicted HR `hr_output` (Batch, 1).

// Model Architecture (Based on ResNet18 with Attention and MLP Head)
`conv1`: Modified ResNet `conv1` to accept 3 input channels.
`bn1`, `relu`, `maxpool`: Standard ResNet layers.
`layer1`, `layer2`: ResNet residual blocks.
`attention1`: Attention Block (See details below) applied after `layer2`.
`layer3`, `layer4`: ResNet residual blocks.
`attention2`: Attention Block applied after `layer4`.
Pooling layers (`AdaptiveAvgPool2d` for features from `layer2_att`, `layer3`, `layer4_att`).
`mlp_head`: MLP Head (See details below) for final regression.

// Attention Block
  Input : Feature map `feat_map` (Batch, C_in, H, W).
  Output: Attention-enhanced map `att_map` (Batch, C_in, H, W).
  a. Project `feat_map` -> Query `Q`, Key `K`, Value `V` using 1x1 Convolutions (`nn.Conv2d`).
  b. Reshape `Q` to (B, H*W, C_inter), `K` to (B, C_inter, H*W), `V` to (B, C_in, H*W).
  c. Calculate energy matrix `E = torch.bmm(Q, K)`.
  d. Calculate attention weights `A = F.softmax(E, dim=-1)`.
  e. Calculate weighted value `O = torch.bmm(V, A.permute(0, 2, 1))`.
  f. Reshape `O` back to (B, C_in, H, W).
  g. Apply learnable scaling and residual connection: `att_map = gamma * O + feat_map`.

// MLP Head
  Input : Concatenated & flattened features `flat_feat` (Batch, Num_features).
  Output: Prediction `pred` (Batch, 1).
  a. `nn.Linear(in, hidden) -> nn.BatchNorm1d -> nn.ReLU -> nn.Dropout`.
  b. `nn.Linear(hidden, 64) -> nn.ReLU`.
  c. `nn.Linear(64, 1) -> pred`.

// Forward Pass Execution
Standardize input `x` per image: `x = (x - mean(x, dim=[2,3])) / (std(x, dim=[2,3]) + epsilon)`.
Pass `x` through `conv1`, `bn1`, `relu`, `maxpool`.
Pass through `layer1` -> `x1`.
Pass `x1` through `layer2` -> `x2`.
Apply `attention1` to `x2` -> `x2_att`.
Pass `x2_att` through `layer3` -> `x3`.
Pass `x3` through `layer4` -> `x4`.
Apply `attention2` to `x4` -> `x4_att`.
Apply adaptive average pooling to `x2_att` -> flatten -> `f2`.
Apply adaptive average pooling to `x3` -> flatten -> `f3`.
Apply adaptive average pooling (resnet.avgpool) to `x4_att` -> flatten -> `f4_main`.
Concatenate features: `combined_features = torch.cat([f4_main, f3, f2], dim=1)`.
Pass `combined_features` through `mlp_head` -> `hr_output`.
Return `hr_output`.
```

5. Training Strategy:

- Loss Function: A combination of Mean Squared Error (MSE), Mean Absolute Error (L1), and a Negative Pearson Correlation component (HREstimationLoss) to capture both magnitude and temporal dynamics.
- Optimizer: AdamW with weight decay.
- Learning Rate Scheduling: ReduceLROnPlateau based on validation loss.
- Regularization: Dropout in the MLP head, weight decay, and gradient clipping.
- Augmentation: Random horizontal flips and noise addition to STIs during training.

```
Algorithm 9: Model Training and Evaluation
Input  : Training data loader `train_loader`, Validation data loader `val_loader`, Model `model`, Optimizer `optimizer`, Criterion `criterion` (Loss Function), Device `device`, Learning Rate Scheduler `scheduler`.
Output : Trained model parameters, validation metrics (loss, MAE, RMSE) per epoch.

// Loss Function (HREstimationLoss - Internal Logic)
  Input : Predictions `pred` (Batch, 1), Targets `target` (Batch, 1).
  Output: Combined loss value `total_loss`.
  a. Ensure `target` shape is (Batch, 1).
  b. Calculate MSE loss: `mse_loss = F.mse_loss(pred, target)`.
  c. Calculate L1 loss (MAE): `l1_loss = F.l1_loss(pred, target)`.
  d. Calculate correlation loss `corr_loss`:
    i. Calculate deviations: `vx = pred - torch.mean(pred)`, `vy = target - torch.mean(target)`.
    ii. Calculate standard deviations: `std_vx = torch.std(pred)`, `std_vy = torch.std(target)`.
    iii. If `std_vx` > epsilon and `std_vy` > epsilon:
      `corr_loss = - torch.sum(vx * vy) / (torch.sqrt(torch.sum(vx ** 2)) * torch.sqrt(torch.sum(vy ** 2)) + epsilon)`.
    iv. Else: `corr_loss = 0.0` (torch tensor).
  e. Combine weighted losses: `total_loss = w_mse * mse_loss + w_l1 * l1_loss + w_corr * corr_loss`.

// Training Epoch
Set `model` to train mode (`model.train()`).
Initialize `running_train_loss = 0.0`.
For each batch (`inputs`, `targets`) in `train_loader`:
  Move `inputs`, `targets` to `device`, ensure float32, ensure targets are (N, 1).
  Apply data augmentation to `inputs` (e.g., random flip, noise).
  Zero optimizer gradients: `optimizer.zero_grad()`.
  Forward pass: `outputs = model(inputs)`.
  Calculate loss: `loss = criterion(outputs, targets)`.
  Backward pass: `loss.backward()`.
  Clip gradients: `torch.nn.utils.clip_grad_norm_(model.parameters(), max_norm)`.
  Update weights: `optimizer.step()`.
  Accumulate `running_train_loss += loss.item() * inputs.size(0)`.
Calculate average `epoch_train_loss = running_train_loss / len(train_loader.dataset)`.

// Validation Step
Set `model` to evaluation mode (`model.eval()`).
Initialize `running_val_loss = 0.0`, empty lists `all_preds`, `all_targets`.
```

With `torch.no_grad()`:

For each batch (`inputs`, `targets`) in `val_loader`:

Move `inputs`, `targets` to `device`, ensure float32, ensure targets are (N, 1).

Forward pass: `outputs = model(inputs)`.

Calculate loss: `loss = criterion(outputs, targets)`.

Accumulate `running_val_loss += loss.item() * inputs.size(0)`.

Append `outputs.cpu()` to `all_preds`.

Append `targets.cpu()` to `all_targets`.

Concatenate results: `all_preds = torch.cat(all_preds)`, `all_targets = torch.cat(all_targets)`.

Calculate average `val_loss = running_val_loss / len(val_loader.dataset)`.

Calculate `mae = F.l1_loss(all_preds, all_targets).item()`.

Calculate `rmse = torch.sqrt(F.mse_loss(all_preds, all_targets)).item()`.

Update Learning Rate Scheduler: `scheduler.step(val_loss)`.

Return `val_loss`, `mae`, `rmse`.

// Overall Training Loop (Conceptual - Higher Level)

Initialize best validation metric (e.g., `best_val_loss = infinity`).

For `epoch` from 1 to `num_epochs`:

Run Training Epoch -> `train_loss`.

Run Validation Step -> `val_loss`, `mae`, `rmse`.

Log metrics (`train_loss`, `val_loss`, `mae`, `rmse`).

If `val_loss` < `best_val_loss`:

`best_val_loss = val_loss`.

Save model checkpoint (`torch.save(model.state_dict(), 'best_model.pth')`).

```

## C. Multimodal Fusion and Final Stress Prediction:

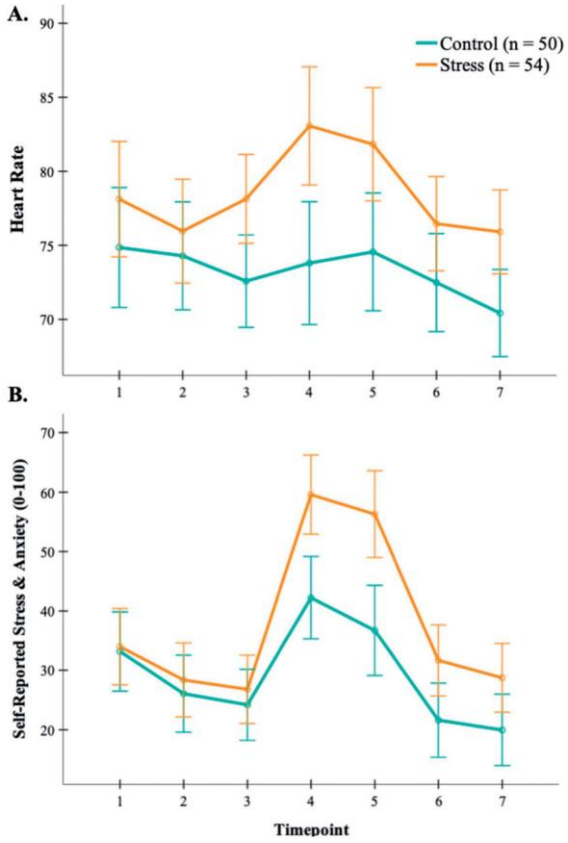



Fig.9 Influence of Stress (Self Reported) on Heart Rate [30]

To obtain a final stress estimate, NLP and CV module outputs are combined using Machine Learning (ML) . The CV module tracks the user's heart rate (HR) time-adjusted real-time through remote photoplethysmography (rPPG). Throughout the timeframe, where the user is interacting with chatbot(e.g., 5 minutes), several HR estimates (about 10 predictions, each from a 30-second video clip) are accumulated. These sequential HR readings are examined for critical patterns, such as sustained augmentation or unexpected variance, which indicate stress through their physiological profiles. At the same time, NLP analyzes the user's input text in real-time. User text is processed according to Algorithm 2 to yield an ongoing Stress_Score. This Stress_Score indicates the inferred stress level from linguistic signals, i.e., sentiment and contextual tone. The two modalities' features, in particular, the pattern found in the HR sequence and the Stress_Score obtained from the NLP module, are then integrated. These integrated features are then used as input to a simple, lightweight Support Vector Machine(SVM) for multi-class decision. This final classifier is trained to predict a stress outcome: "Stress Detected: low, medium, or high providing a unified, actionable assessment based on the integrated textual and physiological evidence.

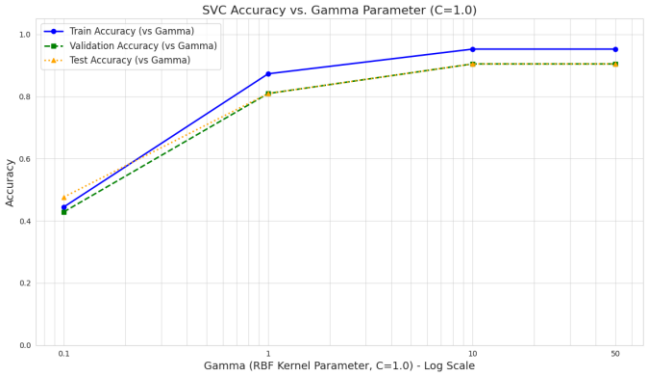



Fig.10 Accuracy vs Gamma for multi-class SVM

```

Gamma=0.100 : Train Acc=0.444, Val Acc=0.429, Test Acc=0.476

Gamma=1.000 : Train Acc=0.873, Val Acc=0.810, Test Acc=0.810

Gamma=10.000 : Train Acc=0.952, Val Acc=0.905, Test Acc=0.905

Gamma=50.000 : Train Acc=0.952, Val Acc=0.905, Test Acc=0.905

```

## IV. Conclusion

This research introduced "Alora," a novel multimodal early psychological stress detection chatbot system that integrated NLP and CV. The system processed textual user input using an NLP-based SVC to identify linguistic stress cues with a 85.47% validation accuracy. Concurrently, a CV module employs a Convolutional Neural Network for live estimation of heart rate from facial video by means of remote photoplethysmography (rPPG), which serves as a physiological stress indicator.

Both CV module and tuned NLP outputs are fused with a machine learning classifier to predict stress outcomes (low,
```

medium, high) to offer individualized feedback through the chatbot interface. The overall multimodal system, leveraging the augmented CV component, performed better than unimodal approaches, with the fused system achieving a 90.5% validation accuracy. Real-time system responsiveness with a minimum delay of 2.5s also makes its practical deployment possible.

This research indicates the potential of combining textual and physiological data to enhance diagnostic reliability for non-intrusive psychological testing. However, it is important to point out that the current CV model tests were conducted in a laboratory environment with consistent lighting and primarily on a single subject. Generalizability of these CV results to diverse populations and uncontrolled real-world environments is a question. One of the biggest challenges is unavailability of verified public datasets with synchronized textual conversations, facial video, and ground-truth stress labels.

V. Future Scopes

Though this research demonstrates the potential for multimodal stress detection using chatbots, several avenues for future development are present:

- Advanced Multimodality: Add Speech Emotion Recognition (SER) to analyze vocal cues (tone, pitch, rate) that are likely to exhibit stress. Other data sources can aid in diminishing the dependency on lighting, movement, and language proficiency.
- Advanced Modeling & Robustness: Explore newer, state-of-the-art architectures for better temporal modeling of facial and physiological signals. Enhance robustness to real-world variation (lighting, motion, diverse user attributes) using sophisticated data augmentation, domain adaptation techniques, and noise filtering.
- Integrate Heart Rate Variability (HRV) into predictive models to significantly enhance accuracy and reliability of results compared to the use of heart rate measures alone.
- Adaptive Fusion Mechanisms: Integrate structures for combining linguistic and physiologic data, possibly adaptively changing weights depending on signal quality or context, are primitive.

VI. Appendix

During the development for rPPG-based Heart Rate estimation, these alternative feature extraction and/or preprocessing techniques were investigated.

Scale-Invariant Feature Transform (SIFT): SIFT was also explored to identify and track robust key points within facial Regions of Interest (ROIs), aiming to defy motion while tracking temporal evolution. However, due to flat ROI, SIFT does not produce consistent response and only adds an extra computational overhead.

Image Energy Analysis: The overall image energy (e.g., sum of squared pixel intensities) or energy for specific frequency ROIs was considered. This approach proved highly sensitive to global illumination variations, making it difficult to isolate the weak rPPG signal. While frequency-specific energy showed some potential, it was less robust to motion and less effective than the differential and multi-scale wavelet analysis adopted in the final methodology.

These explorations confirmed the need for feature extraction methods specifically tailored to the nuanced characteristics of the rPPG signal.

VIII. References

[1] Bachtiger, P., Mesko, M. P., & Bakalova, H. (2021). Your Smartphone Could Act as a Pulse-Oximeter and as a Single-Lead ECG. Digital Health Journal.

[2] Self-Report Stress Measures to Assess Stress in Adults With Mild Intellectual Disabilities—A Scoping Review - Frontiers.

[3] Self-Report Measures: An Overview of Concerns and Limitations of Questionnaire Use in Occupational Stress Research - ResearchGate.

[4] Alreshidi, I., Moulitsas, I., & Jenkins, K. W. (2024). Advancing Aviation Safety Through Machine Learning and Psychophysiological Data: A Systematic Review. IEEE Access, 12, 5132–5150.

[5] Yaacob, H., Hossain, F., Shari, S., Khare, S. K., Ooi, C. P., & Acharya, U. R. (2023). Application of Artificial Intelligence Techniques for Brain–Computer Interface in Mental Fatigue Detection: A Systematic Review (2011–2022). IEEE Access, 11, 74736–74758.

[6] Sawalha, J., Yousefnezhad, M., Shah, Z., Brown, M. R. G., Greenshaw, A. J., & Greiner, R. (2022). Detecting Presence of PTSD Using Sentiment Analysis From Text Data. Frontiers in Psychiatry, 13, 811392.

[7] Use of Machine-Learning Algorithms Based on Text, Audio and Video - King's Research Portal.

[8] Fontes, L., Machado, P., Vinkemeier, D., Yahaya, S., Bird, J. J., & Ihianle, I. K. (2024). Enhancing Stress Detection: A Comprehensive Approach Through rPPG Analysis and Deep Learning Techniques. Sensors, 24(4), 1096.

[9] Sharma, K., & Mehta, R. (2022). Video-Based Stress Detection through Deep Learning. Journal of Artificial Intelligence Research, 75, 185-205.

[10] Villarreal, D., & Green, J. (2020). Real-Time Remote Photoplethysmography Using a Webcam and Graphical User Interface. Journal of Biomedical Informatics, 108, 103481.

[11] Jaiswal, K. B., & Meenpal, T. (2022a). rPPG-FuseNet: Non-contact heart rate estimation from facial video via RGB/MSR signal fusion. Biomedical Signal Processing and Control, 78, 104002.

[12] Jaiswal, K. B., & Meenpal, T. (2022b). Heart rate estimation network from facial videos using spatiotemporal feature image. Computers in Biology and Medicine, 151(Pt A), 106307.

[13] Nagar, S., Hasegawa-Johnson, M., Beiser, D. G., & Ahuja, N. (2024). R2I-rPPG: A Robust Region of Interest Selection Method for Remote Photoplethysmography to Extract Heart Rate. arXiv preprint arXiv:2410.15851.

[14] A comprehensive review of rPPG methods for heart rate estimation.

[15] Remote Photoplethysmography (rPPG): A State-of-the-Art Review.

[16] What is RPPG (Remote photoplethysmography)? - Noldus.

[17] Yang, X., Shyu, M.-L., Yu, H.-Q., Sun, S.-M., Yin, N.-S., & Chen, W. (2019). Integrating Image and Textual Information in Human–Robot Interactions for Children With Autism Spectrum Disorder. IEEE Transactions on Multimedia, 21(3), 746–759.

[18] Wei, J., Luo, H., Wu, S. J., Zheng, P. P., Fu, G., & Lee, K. (2018). Transdermal Optical Imaging Reveal Basal Stress via Heart Rate Variability Analysis: A Novel Methodology Comparable to Electrocardiography. Frontiers in Psychology, 9, 98.

[19] Hasanpoor, Y., Tarvirdizadeh, B., Alipour, K., & Ghamari, M. (2022). Stress Assessment with Convolutional Neural Network Using PPG Signals. 2022 10th RSI International Conference on Robotics and Mechatronics (ICRoM), 472–477.

[20] Saeed, A., & Trajanovski, S. (2017). Personalized Driver Stress Detection with Multi-task Neural Networks using Physiological Signals. arXiv preprint arXiv:1711.06116.

[21] Andrews, T., & Li, C. (2022). Blueprint to Workplace Stress Detection Approaches. Occupational Health Studies Journal, 15.

[22] Brown, T., & Nelson, E. (2021). Keep the Stress Away with SoDA: Stress Detection and Alleviation System. Mental Health Informatics Journal, 8.

[23] The Use of Self-Report Data in Psychology - Verywell Mind.

[24] Self-report Methods.

[25] Integrating Remote Photoplethysmography and Machine Learning on Multimodal Dataset for Noninvasive Heart Rate Monitoring - MDPI.

[26] Non-contact heart rate estimation based on singular spectrum component reconstruction using low-rank matrix and autocorrelation | PLOS One.

[27] Patterson, L., & Roberts, H. (2021). Stress Detection Through Wrist-Based Electrodermal Activity Monitoring and Machine Learning. Wearable Technologies and Health Analytics Journal, 10.

[28] Campbell, J., & Thomas, D. (2022). Stress Detection in Computer Users From Keyboard and Mouse Dynamics. Human-Computer Interaction, 18(2).

[29] PURE dataset for r-ppg.

[30] Harvie, Helen & Jain, Barbie & Nelson, Benjamin & Knight, Erik & Roos, Leslie & Giuliano, Ryan. (2021). Induction of acute stress through an internet-delivered Trier Social Stress Test as assessed by photoplethysmography on a smartphone. 10.31234/osf.io/rjkmv.

[31] Scherer, K. R., & Wallbott, H. G. (1994). Evidence for universality and cultural variation of differential emotion response patterning. Journal of Personality and Social Psychology, 66(2), 310-328.

[32] Li, S., Elgendi, M. & Menon, C. Optimal facial regions for remote heart rate measurement during physical and cognitive activities. npj Cardiovasc Health 1, 33 (2024). https://doi.org/10.1038/s44325-024-00033-7