National College of Ireland

Comparative Analysis of Models to Improve Technical Speaking through AI

MSc Research Project
MSc AI

Ahmed Zeeshan Mazhar
Student ID: x23391880

School of Computing
National College of Ireland

Supervisor: Arundev Vamadevan

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: …….Ahmed zeeshan mazhar……

Student ID: ……x23391880……..……

Programme: ……MSc AI…… **Year:** …2024-25…..

Module: ……Practicum…….……

Supervisor: ……Arundev Vamadevan…….……

Submission Due Date: ……11/08/25…….……

Project Title: Comparative Analysis of Models to Improve Technical Speaking through AI

Word Count: ……1292…… **Page Count**……23…….……..

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: ……

Date: ……

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Forename Surname
Student ID

1 Introduction

The configuration manual explains the hardware and software specifications that are used in this study. This also explains the steps taken to carry out the whole study.

2 Software and System Specifications

We used the Google Colab for whole study implementation tasks and this was implemented on a windows laptop. Figure 1 shows the specifications of the system and windows information is provided in Figure 2.

Device name	DESKTOP-M3AA6DL
Processor	Intel(R) Core(TM) i7-10610U CPU @ 1.80GHz (2.30 GHz)
Installed RAM	32.0 GB (31.7 GB usable)
Device ID	BB2C229A-FD88-4D86-8F4A-E6E50F677674
Product ID	00330-53330-52883-AAOEM
System type	64-bit operating system, x64-based processor
Pen and touch	Touch support with 10 touch points

Figure 1: System Specifications

Edition	Windows 11 Pro
Version	24H2
Installed on	12/13/2024
OS build	26100.4770
Experience	Windows Feature Experience Pack 1000.26100.197.0

Figure 2: Windows Specifications

We used the Google Colab and the following steps are taken to set the environment.

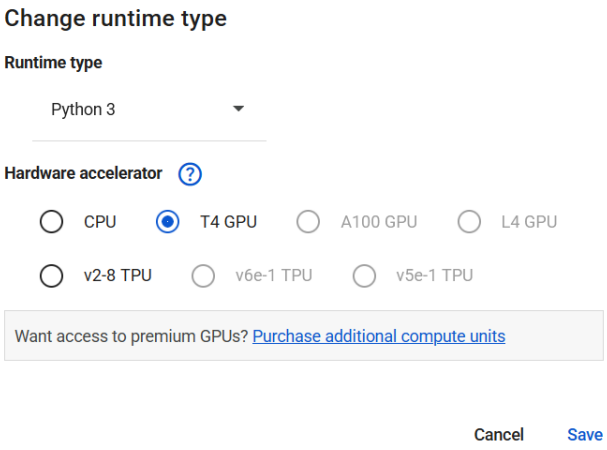

Figure 3: Google Colab Environment

The following resources with memory and disk are offered by T4 GPU.

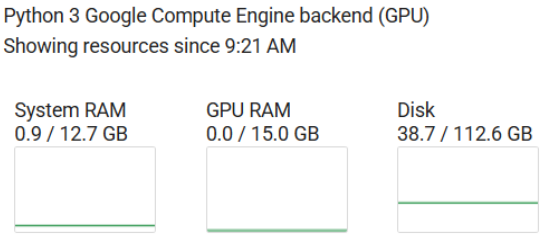

Figure 4: GPU Resources

3 Datasets Description

We use two different datasets for audio and video analysis. We downloaded each dataset during implementation. We downloaded the audio dataset named CMU-MOSEI from Kaggle using KaggleHub (Hồ, 2023). The dataset contains 5,245 audio files. We extract four features from these audio files. These features include words per minute (WPM), complexity scores, bias words, and filler words. For video analysis, we use a dataset obtained from Roboflow (Face recognition, 2023). This dataset contains images with six classes: Angry, Disgust, Happy, Natural, Sad, and Surprise. Each image has labels and bounding boxes. These are necessary for You Only Look Once (YOLO) models.

4 Exploratory Data Analysis (EDA)

We used the audio dataset and performed EDA to understand the dataset. The following steps are taken to perform the EDA.

1. Install the packages

```
!pip install kagglehub torchaudio whisper openai-whisper librosa numpy transformers textstat spacy
```

Figure 5: Packages Installations

2. Audio dataset download

```
import kagglehub

# Download CMU-MOSEI dataset
path = kagglehub.dataset_download("gnurtqh/cmu-mosei")

print("Path to dataset files:", path)
```

Figure 6: Audio Dataset Download

3. Exploring the directory of the data

```
import os

# Recursive directory explorer (limited file samples per folder)
def explore_directory_with_samples(root_dir, sample_limit=5):
    print(f"\nExploring directory: {root_dir}\n{'-'*70}")
    for dirpath, dirnames, filenames in os.walk(root_dir):
        level = dirpath.replace(root_dir, '').count(os.sep)
        indent = ' ' * 4 * level
        print(f"{indent}{os.path.basename(dirpath)}/")

        # Show limited number of sample files from this folder
        subindent = ' ' * 4 * (level + 1)
        sample_files = filenames[:sample_limit]
        for f in sample_files:
            print(f"{subindent}{f}")

        if len(filenames) > sample_limit:
            print(f"{subindent}... and {len(filenames) - sample_limit} more files")

# Run the explorer
explore_directory_with_samples(path)
```

Figure 7: Directory Exploration of Data

4. Audio features definition and creation

```
import os
import torchaudio
import torchaudio.transforms as T

# Path to training audio files
AUDIO_DIR = os.path.join(
    path,
    "CMU-MOSEI-20230514T151450Z-001",
    "CMU-MOSEI",
    "Labels",
    "Audio_chunk",
    "Train_modified"
)

def load_audio(filepath):
    waveform, sample_rate = torchaudio.load(filepath)
    if sample_rate != 16000:
        resampler = T.Resample(orig_freq=sample_rate, new_freq=16000)
        waveform = resampler(waveform)
    return waveform, 16000
```

```
import whisper

model = whisper.load_model("base")  # choose: base, small, medium, large

def transcribe_audio(audio_path):
    result = model.transcribe(audio_path, fp16=False)
    return result['text']
```

```
import spacy
import textstat

filler_words = {'um', 'uh', 'like', 'you know', 'so', 'actually', 'basically', 'literally', 'I mean', 'okay'}

def detect_filler_words(transcript):
    words = transcript.lower().split()
    fillers = [word for word in words if word in filler_words]
    return fillers, len(fillers)

def calculate_wpm(transcript, duration_seconds):
    num_words = len(transcript.split())
    return (num_words / duration_seconds) * 60

nlp = spacy.load("en_core_web_sm")

def analyze_complexity(transcript):
    doc = nlp(transcript)
    num_sentences = len(list(doc.sents))
    flesch_score = textstat.flesch_reading_ease(transcript)
    return {
        "sentence_count": num_sentences,
        "flesch_score": flesch_score,
    }

bias_words = {"guys", "manpower", "chairman", "crazy", "insane"}

def detect_bias_words(transcript):
    words = transcript.lower().split()
    found = [word for word in words if word in bias_words]
    return found
```

```
bias_found = detect_bias_words(transcript)

# Print Output
print("Transcript:\n", transcript)
print("Filler Words:", fillers)
print("Filler Count:", filler_count)
print("Words Per Minute:", round(wpm, 2))
print("Sentence Complexity:", complexity)
print("Bias-related Words:", bias_found)
print("Processing Time:", round(end_time - start_time, 2), "s")
```

Figure 8:Audio Features Creation

5. Distribution of audio durations

```
# Distribution of Audio Durations
plt.figure(figsize=(8, 4))
sns.histplot(df["duration_sec"], bins=15, kde=True)
plt.title("Distribution of Audio Durations")
plt.xlabel("Duration (seconds)")
plt.ylabel("Count")
plt.tight_layout()
plt.show()
```

Figure 9: Audio Duration Distribution

6. Words Per Minute (WPM) distribution

```
# Words Per Minute Distribution
plt.figure(figsize=(8, 4))
sns.histplot(df["wpm"], bins=15, kde=True, color="orange")
plt.title("Words Per Minute Distribution")
plt.xlabel("WPM")
plt.tight_layout()
plt.show()
```

Figure 10: WPM Distribution

7. Complexity score distribution

```
# Flesch Reading Ease Score Distribution
plt.figure(figsize=(8, 4))
sns.histplot(df["flesch_score"], bins=15, kde=True, color="green")
plt.title("Flesch Reading Ease Scores")
plt.xlabel("Score")
plt.tight_layout()
plt.show()
```

Figure 11: Flesch (Complexity) Score Distribution

8. Filler word counts

```
# Filler Word Counts
plt.figure(figsize=(8, 4))
sns.violinplot(x=df["filler_count"], color='lightgreen')
plt.title("Filler Word Counts")
plt.tight_layout()
plt.show()
```

Figure 12: Filler Word Counts

9. Bias word counts

```
# Bias Word Counts
plt.figure(figsize=(8, 4))
sns.countplot(x=df["bias_count"])
plt.title("Bias Word Count per Audio File")
plt.xlabel("Bias Word Count")
plt.tight_layout()
plt.show()
```

Figure 13: Bias Word Counts

10. Top 5 filler words

```
# Top 5 by Filler Words
print("Top 5 Audio Files by Filler Word Count:\n")
df.sort_values(by="filler_count", ascending=False)[["filename", "filler_count", "transcript"]].head()
```

Figure 14: Top 5 Filler Words in Data

11. Highly complex transcripts

```
# Lowest Flesch Score (hardest to read)
print("Top 5 Complex Transcripts (Lowest Flesch Score):\n")
df.sort_values(by="flesch_score")[["filename", "flesch_score", "transcript"]].head()
```

Figure 15: Top 5 Hardest Reading Transcripts

5 Project Development

The project development is divided into three parts. We developed three models. The development and testing of these models are explained using the following implementation.

5.1 Model 1: Extra Trees Regressor and YOLOv8

The Model 1 is developed by first developing Extra Trees (ET) Regressor and YOLO version 8 (YOLOv8) separately and then integrating them.

5.1.1 Extra Trees Regressor

We explain the development and testing of Extra Trees (ET) Regressor in simple steps. Following are the key steps taken.

1. **Installing packages and importing the libraries**

```
!pip install kagglehub torchaudio whisper openai-whisper librosa numpy transformers textstat spacy

import os
import joblib
import numpy as np
import librosa
import whisper
import torchaudio
import spacy
import re
import textstat
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.multioutput import MultiOutputRegressor
from sklearn.metrics import (
    mean_squared_error,
    mean_absolute_error,
    r2_score,
    explained_variance_score,
    max_error
)
from sklearn.ensemble import ExtraTreesRegressor
```

Figure 16: Package Installation and Importing Libraries

2. **Audio dataset download from Kaggle**

```python
import kagglehub

# Download CMU-MOSEI dataset
path = kagglehub.dataset_download("gnurtqh/cmu-mosei")

print("Path to dataset files:", path)
```

Figure 17: Audio Dataset Download

3. Save the preprocessed data in notebook

```python
import numpy as np
import os

# Create a directory to save them
os.makedirs("Audio preprocessed data", exist_ok=True)

# Save feature and label arrays
np.save("Audio preprocessed data/X_features.npy", X)
np.save("Audio preprocessed data/y_labels.npy", y)

print("Saved X and y as .npy files.")
```

Figure 18: Save the Preprocessed Data

4. Save the preprocessed data in google drive

```python
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```python
import shutil

# Source and destination paths
source_folder = "/content/Audio preprocessed data"
destination_folder = "/content/drive/MyDrive/Audio preprocessed data in text"

# Copy the folder and its contents
shutil.copytree(source_folder, destination_folder)

print("Folder copied successfully to Drive at:", destination_folder)
```

Figure 19: Save the Data in Google Drive

5. Load the data and split into training and testing

```
X = np.load("/content/drive/MyDrive/Audio preprocessed data in text/X_features.npy")
y = np.load("/content/drive/MyDrive/Audio preprocessed data in text/y_labels.npy")
```

```
# Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Scale
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Target Names
target_names = ['Filler Words', 'Words per Minute', 'Complexity Score', 'Bias Words']
```

Figure 20: Load the Preprocessed Dataset and Split

6. ET model development and testing

```
from sklearn.ensemble import ExtraTreesRegressor

model = MultiOutputRegressor(ExtraTreesRegressor(
    n_estimators=100,
    max_depth=15,
    random_state=42,
    n_jobs=-1
))
model.fit(X_train_scaled, y_train)
y_pred = model.predict(X_test_scaled)

print("Multi-Output ExtraTrees Evaluation")
for i, name in enumerate(target_names):
    print(f"\n{name}:")
    print(f"MSE                : {mean_squared_error(y_test[:, i], y_pred[:, i]):.2f}")
    print(f"MAE                : {mean_absolute_error(y_test[:, i], y_pred[:, i]):.2f}")
    print(f"R² Score           : {r2_score(y_test[:, i], y_pred[:, i]):.2f}")
    print(f"Explained Variance : {explained_variance_score(y_test[:, i], y_pred[:, i]):.2f}")
    print(f"Max Error          : {max_error(y_test[:, i], y_pred[:, i]):.2f}")

os.makedirs("Audio_saved_model_2", exist_ok=True)
joblib.dump(model, "Audio_saved_model_2/multi_et_model.pkl")
joblib.dump(scaler, "Audio_saved_model_2/scaler_et.pkl")
print("\nModel and Scaler Saved")
```

Figure 21: ET Model Development

7. Save the ET model in google drive for future use

```
import shutil

# Source and destination paths
source_folder = "/content/Audio_saved_model_2"
destination_folder = "/content/drive/MyDrive/Audio_saved_model_2"

# Copy the folder and its contents
shutil.copytree(source_folder, destination_folder)

print("Folder copied successfully to Drive at:", destination_folder)
```

Figure 22: Save ET Model in Google Drive

5.1.2 YOLOv8 Development

The following steps are taken to develop YOLOv8.

1. **Package installation and importing libraries**

```
!pip install roboflow ultralytics

import ultralytics
from roboflow import Roboflow
import cv2
import os
import glob
import matplotlib.pyplot as plt
import seaborn as sns
from tqdm import tqdm
from ultralytics import YOLO
from IPython.display import Image, display
```

Figure 23: Packages Installation and Importing Libraries

2. **Dataset download and move to a directory**

```
from roboflow import Roboflow
rf = Roboflow(api_key="2omg7mQQrWQC7v9SNLvs")
project = rf.workspace("face-recognition-ixqtg").project("emotion-detection-cwq4g")
version = project.version(1)
dataset = version.download("yolov11")
```

```
loading Roboflow workspace...
loading Roboflow project...
Downloading Dataset Version Zip in emotion-detection-1 to yolov11:: 100%|████████| 1440150/1440150 [00:22<00:00, 62680.18it/s]

Extracting Dataset Version Zip to emotion-detection-1 in yolov11:: 100%|████████| 42538/42538 [00:09<00:00, 4725.12it/s]
```

```
!mv /content/emotion-detection-1 /content/data
```

Figure 24: Images Data Download

3. **Images resizing for easy computation and enhancing accuracy**

```
# Define paths and resize images
image_paths = ['/content/data/train/images', '/content/data/valid/images', '/content/data/test/images']
target_size = (640, 640)
for path in image_paths:
    for img_file in tqdm(os.listdir(path)):
        img_path = os.path.join(path, img_file)
        img = cv2.imread(img_path)
        if img is not None:
            resized_img = cv2.resize(img, target_size)
            cv2.imwrite(img_path, resized_img)
print("\nImage resizing complete!")
```

Figure 25: Images Resizing

4. **YOLOv8 Training**

```
# Train YOLOv8 with Early Stopping
model = YOLO('yolov8n.pt')
model.train(data='/content/data/data.yaml', epochs=30, imgsz=640, batch=16, name='yolo_facial_v8')
```

Figure 26: YOLOv8 Training

5. **YOLOv8 Evaluating**

```
# Load best model
model = YOLO('/content/runs/detect/yolo_facial_v8/weights/best.pt')

# Run validation on validation set
metrics = model.val()
print(metrics)  # Includes mAP, precision, recall, etc.
```

Figure 27: YOLOv8 Evaluating

6. Save YOLOv8 in notebook

```
from ultralytics import YOLO

# Load the trained YOLO model
model = YOLO('/content/runs/detect/yolo_facial_v8/weights/best.pt')

# Export to ONNX
model.export(format='onnx')
```

Figure 28: Save YOLOv8

7. Save YOLOv8 in google drive

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
import shutil

src_path = '/content/runs/detect/yolo_facial_v8/weights/best.onnx'
dst_path = '/content/drive/MyDrive/yolov8_best_facial_expression.onnx'

shutil.copy(src_path, dst_path)
print("ONNX model copied to Google Drive successfully.")
```

Figure 29: Save YOLOv8 in Google Drive

8. YOLOv8 testing results

```
import glob
from IPython.display import Image, display

# Inference and results
model = YOLO('/content/runs/detect/yolo_facial_v8/weights/best.pt')
results = model.predict(source='/content/data/test/images', save=True, save_txt=True, conf=0.5)
result_images = glob.glob('runs/detect/predict/*.jpg')
for img in result_images[:20]:
    display(Image(filename=img))
```

Figure 30: YOLOv8 Testing Results

9. YOLOv8 inference time computation

```
import time

start_time = time.time()
model.predict(source='/content/data/test/images', save=False)
end_time = time.time()

fps = len(os.listdir('/content/data/test/images')) / (end_time - start_time)
print(f"Inference Speed: {fps:.2f} FPS")
```

Figure 31: YOLOv8 Inference Time Computation

5.1.3 Integrating ET Regressor and YOLOv8

We integrated the ET Regressor and YOLOv8 to create the Model 1. We recorded a video using windows camera to test the Model 1. We used the following steps to test the Model 1.

1. **Importing drive and installing necessary packages**

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
!pip install textstat SpeechRecognition pydub ffmpeg ultralytics onnxruntime
```

2. **Using recorded video to detect facial expressions and extract audio**

```
import cv2
import os
from ultralytics import YOLO
from pydub import AudioSegment
import moviepy.editor as mpe

# Paths
onnx_model_path = '/content/drive/MyDrive/yolov8_best_facial_expression
video_path = '/content/recorded_video.mp4'
frames_dir = '/content/model 1 frames'
output_video_path = '/content/model 1 output_video_no_audio.mp4'
final_video_path = '/content/model 1 final_video_with_audio.mp4'
audio_path = '/content/model 1 extracted_audio.wav'

# Load YOLO model
yolo_model = YOLO(onnx_model_path)

# Create directory for frames
os.makedirs(frames_dir, exist_ok=True)

# Extract frames from input video
cap = cv2.VideoCapture(video_path)
frame_paths = []
count = 0

while True:
    ret, frame = cap.read()
    if not ret:
        break
    resized = cv2.resize(frame, (640, 640))  # Resize for YOLO input
    path = os.path.join(frames_dir, f"frame_{count:03d}.jpg")
    cv2.imwrite(path, resized)
    frame_paths.append(path)
    count += 1
```

```
cap.release()
print(f"Extracted {count} frames.")

# Annotate frames using YOLO
for path in frame_paths:
    img = cv2.imread(path)
    results = yolo_model.predict(source=img, imgsz=640, conf=0.15)[0]
    for box in results.boxes:
        x1, y1, x2, y2 = map(int, box.xyxy[0])
        conf = float(box.conf[0])
        cls_id = int(box.cls[0])
        label = yolo_model.names[cls_id]
        cv2.rectangle(img, (x1, y1), (x2, y2), (0, 255, 0), 2)
        cv2.putText(img, f"{label} {conf:.2f}", (x1, y1 - 10),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 0, 255), 2)
    cv2.imwrite(path, img)

# Create output video from annotated frames
frame_example = cv2.imread(frame_paths[0])
height, width, layers = frame_example.shape
fps = 30

out = cv2.VideoWriter(output_video_path, cv2.VideoWriter_fourcc(*'mp4v'), fps, (width, height))
for path in frame_paths:
    frame = cv2.imread(path)
    out.write(frame)
out.release()
print(f"Annotated video saved to: {output_video_path}")

# Extract audio from original video
audio = AudioSegment.from_file(video_path, format="mp4")
audio.export(audio_path, format="wav")
print(f"Audio extracted to: {audio_path}")
```

```
# Combine annotated video with extracted audio using MoviePy
video_clip = mpe.VideoFileClip(output_video_path)
audio_clip = mpe.AudioFileClip(audio_path)
final_clip = video_clip.set_audio(audio_clip)
final_clip.write_videofile(final_video_path, codec='libx264', audio_codec='aac')
print(f"Final video with audio saved to: {final_video_path}")
```

Figure 32: Video Preprocessing and Facial Expression Detection using YOLOv8

3. Plotting pictures with facial expressions for each second of the video

```python
import cv2
import matplotlib.pyplot as plt

# Load the video
video_path = '/content/model 1 output_video_no_audio.mp4'
cap = cv2.VideoCapture(video_path)

# Get video properties
fps = cap.get(cv2.CAP_PROP_FPS)
total_frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
duration_seconds = int(total_frames / fps)

print(f"FPS: {fps}, Total Frames: {total_frames}, Duration (s): {duration_seconds}")

# Extract and plot 1 frame per second for 14 seconds
for sec in range(1, 15):  # 1 to 14
    cap.set(cv2.CAP_PROP_POS_MSEC, sec * 1000)  # Position in milliseconds
    ret, frame = cap.read()
    if not ret:
        print(f"Frame at second {sec} not found.")
        continue

    # Convert BGR to RGB
    frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

    # Plot the frame
    plt.figure(figsize=(6, 4))
    plt.imshow(frame_rgb)
    plt.title(f"Second {sec}")
    plt.axis('off')
    plt.show()

# Release the video capture
cap.release()
```

Figure 33: Facial Expressions Results Plotting

5.2 Model 2: Random Forest Regressor + YOLOv8

We explain the development of Model 2 in simple steps. Model 2 is created by combining the development of Random Forest (RF) Regressor and YOLOv8.

5.2.1 Random Forest Regressor

We preprocessed the audio data for the Model 1 development and saved it in Google Drive. So, we use this saved dataset instead of again preprocessing. The following steps are taken to develop RF Regressor.

1. **Load the preprocessed data from google drive and split it into training and testing**

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
X = np.load("/content/drive/MyDrive/Audio preprocessed data in text/X_features.npy")
y = np.load("/content/drive/MyDrive/Audio preprocessed data in text/y_labels.npy")
```

```
# Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Scale
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Target Names
target_names = ['Filler Words', 'Words per Minute', 'Complexity Score', 'Bias Words']
```

Figure 34:Load the Preprocessed Data and Split

2. RF Regressor development and testing

```
from sklearn.ensemble import RandomForestRegressor

# Define Random Forest with your given hyperparameters
rf = RandomForestRegressor(
    n_estimators=200,
    max_depth=20,
    min_samples_split=2,
    min_samples_leaf=1,
    random_state=42,
    n_jobs=-1
)

model = MultiOutputRegressor(rf)
model.fit(X_train_scaled, y_train)
y_pred = model.predict(X_test_scaled)

# Evaluation
print("Multi-Output RandomForest Evaluation")
for i, name in enumerate(target_names):
    print(f"\n{name}:")
    print(f"MSE                : {mean_squared_error(y_test[:, i], y_pred[:, i]):.2f}")
    print(f"MAE                : {mean_absolute_error(y_test[:, i], y_pred[:, i]):.2f}")
    print(f"R² Score           : {r2_score(y_test[:, i], y_pred[:, i]):.2f}")
    print(f"Explained Variance : {explained_variance_score(y_test[:, i], y_pred[:, i]):.2f}")
    print(f"Max Error          : {max_error(y_test[:, i], y_pred[:, i]):.2f}")

# Save
os.makedirs("Audio_saved_model_rf", exist_ok=True)
joblib.dump(model, "Audio_saved_model_rf/multi_rf_model.pkl")
joblib.dump(scaler, "Audio_saved_model_rf/scaler_rf.pkl")
print("\nModel and Scaler Saved")
```

Figure 35: RF Regressor Development

3. Save the trained RF Regressor into drive for future use

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
import shutil

# Source and destination paths
source_folder = "/content/Audio_saved_model_rf"
destination_folder = "/content/drive/MyDrive/Audio_saved_model_rf"

# Copy the folder and its contents
shutil.copytree(source_folder, destination_folder)

print("Folder copied successfully to Drive at:", destination_folder)
```

Figure 36: Save the RF Regressor into Google Drive

5.2.2 YOLOv8 Development

YOLOv8 is already trained and saved into the Google Drive. So, we use the saved YOLOv8 instead of training again.

5.2.3 Integrating RF Regressor and YOLOv8

We integrate the RF Regressor and YOLOv8 to develop Model 2. The prerecorded video is again used to test the effectiveness of the Model 2. The following steps are used to test the Model 2.

1. **Using recorded video to detect facial expressions and extract audio**

```python
import cv2
import os
from ultralytics import YOLO
from pydub import AudioSegment
import moviepy.editor as mpe

# Paths
onnx_model_path = '/content/drive/MyDrive/yolov8_best_facial_expression.onnx'
video_path = '/content/recorded_video.mp4'
frames_dir = '/content/model 2 frames'
output_video_path = '/content/model 2 output_video_no_audio.mp4'
final_video_path = '/content/model 2 final_video_with_audio.mp4'
audio_path = '/content/model 2 extracted_audio.wav'

# Load YOLO model
yolo_model = YOLO(onnx_model_path)

# Create directory for frames
os.makedirs(frames_dir, exist_ok=True)

# Extract frames from input video
cap = cv2.VideoCapture(video_path)
frame_paths = []
count = 0

while True:
    ret, frame = cap.read()
    if not ret:
        break
    resized = cv2.resize(frame, (640, 640))  # Resize for YOLO input
    path = os.path.join(frames_dir, f"frame_{count:03d}.jpg")
    cv2.imwrite(path, resized)
    frame_paths.append(path)
    count += 1
cap.release()
print(f"Extracted {count} frames.")

# Annotate frames using YOLO
for path in frame_paths:
    img = cv2.imread(path)
    results = yolo_model.predict(source=img, imgsz=640, conf=0.15)[0]
    for box in results.boxes:
        x1, y1, x2, y2 = map(int, box.xyxy[0])
        conf = float(box.conf[0])
        cls_id = int(box.cls[0])
        label = yolo_model.names[cls_id]
        cv2.rectangle(img, (x1, y1), (x2, y2), (0, 255, 0), 2)
        cv2.putText(img, f"{label} {conf:.2f}", (x1, y1 - 10),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 0, 255), 2)
    cv2.imwrite(path, img)

# Create output video from annotated frames
frame_example = cv2.imread(frame_paths[0])
height, width, layers = frame_example.shape
fps = 30

out = cv2.VideoWriter(output_video_path, cv2.VideoWriter_fourcc(*'mp4v'), fps, (width, height))
for path in frame_paths:
    frame = cv2.imread(path)
    out.write(frame)
out.release()
print(f"Annotated video saved to: {output_video_path}")

# Extract audio from original video
audio = AudioSegment.from_file(video_path, format="mp4")
audio.export(audio_path, format="wav")
print(f"Audio extracted to: {audio_path}")
```

```
# Combine annotated video with extracted audio using MoviePy
video_clip = mpe.VideoFileClip(output_video_path)
audio_clip = mpe.AudioFileClip(audio_path)
final_clip = video_clip.set_audio(audio_clip)
final_clip.write_videofile(final_video_path, codec='libx264', audio_codec='aac')
print(f"Final video with audio saved to: {final_video_path}")
```

Figure 37: Video Preprocessing and Facial Expression Detection using YOLOv8

2. Preprocess the audio and detect different speech features

```
import librosa
import numpy as np
import textstat
import joblib
import whisper
import re
import spacy

# === Load saved scaler and model ===
feature_scaler_path = '/content/drive/MyDrive/Audio_saved_model_rf/scaler_rf.pkl'
multi_output_model_path = '/content/drive/MyDrive/Audio_saved_model_rf/multi_rf_model.pkl'

scaler = joblib.load(feature_scaler_path)
model = joblib.load(multi_output_model_path)

# === Load whisper and spaCy ===
whisper_model = whisper.load_model("base")
nlp = spacy.load("en_core_web_sm")

# === Word sets ===
filler_words = {'um', 'uh', 'like', 'you know', 'so', 'actually', 'basically', 'literally', 'I mean', 'okay'}
bias_words = {"guys", "manpower", "chairman", "crazy", "insane"}

# === Transcription ===
def transcribe_audio(audio_path):
    result = whisper_model.transcribe(audio_path, fp16=False)
    return result['text']

def clean_transcript(text):
    return re.sub(r'[^\w\s]', '', text).lower()

def calculate_wpm(text, duration):
    if duration < 1.0:
        duration = 1.0
```

```
    clean_text = clean_transcript(text)
    words = clean_text.split()
    if len(words) < 3:
        return 0
    return (len(words) / duration) * 60

def detect_filler_words(text):
    words = text.lower().split()
    return [w for w in words if w in filler_words]

def detect_bias_words(text):
    words = text.lower().split()
    return [w for w in words if w in bias_words]

def analyze_complexity(text):
    doc = nlp(text)
    return textstat.flesch_reading_ease(text)

def extract_text_features(transcript):
    words = transcript.split()
    avg_word_len = np.mean([len(w) for w in words]) if words else 0
    num_words = len(words)
    filler_count = len(detect_filler_words(transcript))
    bias_count = len(detect_bias_words(transcript))
    complexity_score = analyze_complexity(transcript)
    return [num_words, avg_word_len, filler_count, complexity_score, bias_count]

# === Audio feature extraction ===
def extract_audio_features(audio_path):
    y, sr = librosa.load(audio_path, sr=16000)
    mfcc = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=13)
    duration = len(y) / sr
    return np.concatenate([np.mean(mfcc, axis=1), np.std(mfcc, axis=1)]), duration
```

```
# === Main test function ===
def test_recorded_audio(file_path):
    audio_feat, duration = extract_audio_features(file_path)
    transcript = transcribe_audio(file_path)
    print(f"\nTranscript:\n{transcript}")

    text_feat = extract_text_features(transcript)
    full_features = np.concatenate([audio_feat, text_feat])
    feat_scaled = scaler.transform([full_features])

    # NLP Analysis Output
    fillers = detect_filler_words(transcript)
    wpm = calculate_wpm(transcript, duration)
    complexity = analyze_complexity(transcript)
    bias_found = detect_bias_words(transcript)

    print(f"\nNLP Analysis:")
    print(f"Filler Words  : {len(fillers)} ({fillers})")
    print(f"Words/Minute  : {wpm:.2f}")
    print(f"Flesch Score  : {complexity:.2f}")
    print(f"Bias Words    : {len(bias_found)} ({bias_found})")

    # Model Prediction
    prediction = model.predict(feat_scaled)[0]
    print(f"\nModel Prediction:")
    print(f"Filler Words     : {round(prediction[0], 2)}")
    print(f"WPM              : {round(prediction[1], 2)}")
    print(f"Complexity Score : {round(prediction[2], 2)}")
    print(f"Bias Words       : {round(prediction[3], 2)}")

test_recorded_audio(audio_path)
```

Figure 38: Preprocessing Audio and Speech Features Detection using RF Regressor

3. Plotting pictures with facial expressions for each second of the video

```python
import cv2
import matplotlib.pyplot as plt

# Load the video
video_path = '/content/model 2 output_video_no_audio.mp4'
cap = cv2.VideoCapture(video_path)

# Get video properties
fps = cap.get(cv2.CAP_PROP_FPS)
total_frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
duration_seconds = int(total_frames / fps)

print(f"FPS: {fps}, Total Frames: {total_frames}, Duration (s): {duration_seconds}")

# Extract and plot 1 frame per second for 14 seconds
for sec in range(1, 15):  # 1 to 14
    cap.set(cv2.CAP_PROP_POS_MSEC, sec * 1000)  # Position in milliseconds
    ret, frame = cap.read()
    if not ret:
        print(f"Frame at second {sec} not found.")
        continue

    # Convert BGR to RGB
    frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

    # Plot the frame
    plt.figure(figsize=(6, 4))
    plt.imshow(frame_rgb)
    plt.title(f"Second {sec}")
    plt.axis('off')
    plt.show()

# Release the video capture
cap.release()
```

Figure 39: Facial Expressions Results Plotting

5.3 Model 3: Random Forest Regressor and YOLOv11

We explain the development of Model 3 in simple steps. Model 3 is created by combining the development of RF Regressor and YOLO version 11 (YOLOv11).

5.3.1 Random Forest Regressor

RF Regressor is already trained and saved. So, we use the saved model instead of training again.

5.3.2 YOLOv11 Development

YOLOv11 is developed using the following key steps.

1. **Dataset download and move to a directory**

```
from roboflow import Roboflow
rf = Roboflow(api_key="2omg7mQQrWQC7v9SNLvs")
project = rf.workspace("face-recognition-ixqtg").project("emotion-detection-cwq4g")
version = project.version(1)
dataset = version.download("yolov11")
```

```
loading Roboflow workspace...
loading Roboflow project...
Downloading Dataset Version Zip in emotion-detection-1 to yolov11:: 100%|██████████| 1440150/1440150 [00:22<00:00, 62680.18it/s]

Extracting Dataset Version Zip to emotion-detection-1 in yolov11:: 100%|██████████| 42538/42538 [00:09<00:00, 4725.12it/s]
```

```
!mv /content/emotion-detection-1 /content/data
```

Figure 40: Images Data Download

2. Images resizing for easy computation and enhancing accuracy

```
# Define paths and resize images
image_paths = ['/content/data/train/images', '/content/data/valid/images', '/content/data/test/images']
target_size = (640, 640)
for path in image_paths:
    for img_file in tqdm(os.listdir(path)):
        img_path = os.path.join(path, img_file)
        img = cv2.imread(img_path)
        if img is not None:
            resized_img = cv2.resize(img, target_size)
            cv2.imwrite(img_path, resized_img)
print("\nImage resizing complete!")
```

Figure 41: Images Resizing

3. YOLOv11 Training

```
# Train YOLOv11 with Early Stopping
model = YOLO('yolo11n.pt')
model.train(data='/content/data/data.yaml', epochs=30, imgsz=640, batch=16, name='yolo_facial_v11')
```

Figure 42: YOLOv11 Training

4. YOLOv11 Evaluation

```
# Load best model
model = YOLO('/content/runs/detect/yolo_facial_v11/weights/best.pt')

# Run validation on validation set
metrics = model.val()
print(metrics)  # Includes mAP, precision, recall, etc.
```

Figure 43: YOLOv11 Evaluation

5. Save the YOLOv11 in notebook

```
from ultralytics import YOLO

# Load the trained YOLO model
model = YOLO('/content/runs/detect/yolo_facial_v11/weights/best.pt')

# Export to ONNX (no 'path' argument allowed)
model.export(format='onnx')
```

Figure 44: Save YOLOv11

6. **Save the YOLOv11 in google drive**

```
import shutil

src_path = '/content/runs/detect/yolo_facial_v11/weights/best.onnx'
dst_path = '/content/drive/MyDrive/yolo_best_facial_expression.onnx'

shutil.copy(src_path, dst_path)
print("ONNX model copied to Google Drive successfully.")
```

Figure 45: Save YOLOv11 in Google Drive

7. **YOLOv11 testing results**

```
import glob
from IPython.display import Image, display

# Inference and results
model = YOLO('/content/runs/detect/yolo_facial_v11/weights/best.pt')
results = model.predict(source='/content/data/test/images', save=True, save_txt=True, conf=0.5)
result_images = glob.glob('runs/detect/predict/*.jpg')
for img in result_images[:20]:
    display(Image(filename=img))
```

Figure 46: Facial Expression Detection Results by YOLOv11

8. **YOLOv11 inference time computation**

```
import time

start_time = time.time()
model.predict(source='/content/data/test/images', save=False)
end_time = time.time()

fps = len(os.listdir('/content/data/test/images')) / (end_time - start_time)
print(f"Inference Speed: {fps:.2f} FPS")
```

Figure 47: YOLOv11 Inference Time Calculation

5.3.3 Integrating RF Regressor and YOLOv11

The RF Regressor and YOLOv11 are integrated to develop Model 3. We use the prerecorded video to test the Model 3 using the following steps.

1. **Using recorded video to detect facial expressions and extract audio**

```python
import cv2
import os
from ultralytics import YOLO
from pydub import AudioSegment
import moviepy.editor as mpe

# Paths
onnx_model_path = '/content/drive/MyDrive/yolo_best_facial_expression.onnx'
video_path = '/content/recorded_video.mp4'
frames_dir = '/content/model 3 frames'
output_video_path = '/content/model 3 output_video_no_audio.mp4'
final_video_path = '/content/model 3 final_video_with_audio.mp4'
audio_path = '/content/model 3 extracted_audio.wav'

# Load YOLO model
yolo_model = YOLO(onnx_model_path)

# Create directory for frames
os.makedirs(frames_dir, exist_ok=True)

# Extract frames from input video
cap = cv2.VideoCapture(video_path)
frame_paths = []
count = 0

while True:
    ret, frame = cap.read()
    if not ret:
        break
    resized = cv2.resize(frame, (640, 640))  # Resize for YOLO input
    path = os.path.join(frames_dir, f"frame_{count:03d}.jpg")
    cv2.imwrite(path, resized)
    frame_paths.append(path)
    count += 1
```

```python
cap.release()
print(f"Extracted {count} frames.")

# Annotate frames using YOLO
for path in frame_paths:
    img = cv2.imread(path)
    results = yolo_model.predict(source=img, imgsz=640, conf=0.15)[0]
    for box in results.boxes:
        x1, y1, x2, y2 = map(int, box.xyxy[0])
        conf = float(box.conf[0])
        cls_id = int(box.cls[0])
        label = yolo_model.names[cls_id]
        cv2.rectangle(img, (x1, y1), (x2, y2), (0, 255, 0), 2)
        cv2.putText(img, f"{label} {conf:.2f}", (x1, y1 - 10),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 0, 255), 2)
    cv2.imwrite(path, img)

# Create output video from annotated frames
frame_example = cv2.imread(frame_paths[0])
height, width, layers = frame_example.shape
fps = 30

out = cv2.VideoWriter(output_video_path, cv2.VideoWriter_fourcc(*'mp4v'), fps, (width, height))
for path in frame_paths:
    frame = cv2.imread(path)
    out.write(frame)
out.release()
print(f"Annotated video saved to: {output_video_path}")

# Extract audio from original video
audio = AudioSegment.from_file(video_path, format="mp4")
audio.export(audio_path, format="wav")
print(f"Audio extracted to: {audio_path}")
```

```
# Combine annotated video with extracted audio using MoviePy
video_clip = mpe.VideoFileClip(output_video_path)
audio_clip = mpe.AudioFileClip(audio_path)
final_clip = video_clip.set_audio(audio_clip)
final_clip.write_videofile(final_video_path, codec='libx264', audio_codec='aac')
print(f"Final video with audio saved to: {final_video_path}")
```

Figure 48: Video Preprocessing and Facial Expression Detection using YOLOv11

2. Plotting pictures with facial expressions for each second of the video

```
import cv2
import matplotlib.pyplot as plt

# Load the video
video_path = '/content/model 3 output_video_no_audio.mp4'
cap = cv2.VideoCapture(video_path)

# Get video properties
fps = cap.get(cv2.CAP_PROP_FPS)
total_frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
duration_seconds = int(total_frames / fps)

print(f"FPS: {fps}, Total Frames: {total_frames}, Duration (s): {duration_seconds}")

# Extract and plot 1 frame per second for 14 seconds
for sec in range(1, 15):  # 1 to 14
    cap.set(cv2.CAP_PROP_POS_MSEC, sec * 1000)  # Position in milliseconds
    ret, frame = cap.read()
    if not ret:
        print(f"Frame at second {sec} not found.")
        continue

    # Convert BGR to RGB
    frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

    # Plot the frame
    plt.figure(figsize=(6, 4))
    plt.imshow(frame_rgb)
    plt.title(f"Second {sec}")
    plt.axis('off')
    plt.show()

# Release the video capture
cap.release()
```

Figure 49: Facial Expressions Results Plotting