Below is the revised Table of Contents, restructured to match the ICS-GP template while incorporating **all** of your project‐specific points:

**Abstract**

The rapid shift to virtual interviews has created an urgent need for scalable, objective evaluation tools. We present an AI-driven video-interview platform that integrates two tightly coupled analysis pipelines, Speech Emotion Recognition (SER) and automated answer evaluation into a single reporting framework.

The SER pipeline aggregates and preprocesses audio from RAVDESS, CREMA-D, TESS, and SAVEE, extracting MFCCs, Mel-spectrograms, zero-crossing rates, and RMS energy, then applies data augmentation. A one-dimensional CNN with batch normalization, pooling, and dropout is trained on stratified splits, achieving 95.1% test accuracy across eight emotions (macro F1=0.96). Interpretability is provided via learning curve and confusion-matrix visualizations.

The answer-evaluation pipeline transcribes responses with a state-of-the-art speech-to-text model, then scores each answer using FAISS-backed retrieval and a multi-stage LLM (GPT-4o) rubric. Exact matches with prior Q/A pairs above 70% trigger a 70:30 blend of historical and fresh scores; otherwise, we average top 3 neighbor scores for a 30:70 blend, or default to a full LLM (GPT-4o) assessment. Rubric criteria are each prompted three times (3 LLM calls), averaged and the rationales are summarized for per-criterion scores, and then combined into a final score.

A unified reporting module merges emotional and content metrics into an interactive PDF/HTML report. Experiments on held-out responses demonstrate that combining historical data with live rubric evaluations enhances scoring consistency and fairness. Its modular design supports future extensions in multimodal candidate assessment.

Turnitin Report  
Acknowledgements  
Table of Contents  
List of Figures  
List of Tables

# 1 Introduction

 1.1 Overview

Candidate assessment and evaluation have traditionally involved extensive manual/human effort, often requiring multiple rounds of interviews, substantial recruiter time, and subjective evaluations to ensure the process’s accuracy and fairness. Inefficiencies, inconsistencies, and biases in evaluating candidates objectively can be challenges of the mentioned traditional approach [1]. Furthermore, increasing numbers of applicants make the traditional methods struggle more, making them insufficient for the growing demand for streamlined, unbiased, and accurate evaluation processes.

Opportunities have emerged to automate and significantly enhance candidate assessment processes with the rapid spread of artificial intelligence (AI). Scalability, fairness, and efficiency can be improved through integrating AI into recruitment [1]. Automated systems using AI can process vast amounts of data rapidly and consistently, thereby enhancing the fairness of evaluations and minimizing human biases [1].

One of the critical areas in AI-driven recruitment is Speech Emotion Recognition (SER). The underlying technology of SER provides quantitative analysis of the emotional state of the candidates in terms of confidence, stress, and enthusiasm—factors that are essential indicators of the appropriateness and effectiveness of an applicant in the respective positions to be assigned to them [2]. Based on speech descriptors like patterns, pitch, and tone, and other acoustic characteristics, the use of SER systems can provide deep insight into emotional and communication abilities and enrich traditional measures with more detailed and accurate candidate profiles [6][7].

In addition, the evaluation of the relevance and correctness of the candidates' responses is another significant challenge. Traditional evaluations relying solely on human interviewers often reflect inherent bias, unequal levels of expertise, and uneven scoring habits. Developments in natural language processing (NLP) and Large Language Models (LLMs) in recent times have overcome these challenges. Models like BERT, GPT, and LLaMA offer high capabilities in accurately understanding, contextualizing, and valuing linguistic information can be achieved through [16][17][31]. The models can also be augmented with the use of Retrieval-Augmented Generation (RAG), which combines the generative capabilities of LLMs and retrieval of factual information to enable more accurate and contextually relevant assessments of the candidates [18][29].

Moreover, automated reporting significantly enhances recruitment efficiency. Automated systems effectively reduce the time-consuming manual documentation required in the assessment of candidates, thus enabling recruiters to be more strategically focused on decision-making and on engaging with candidates rather than tedious administration tasks [12][13][14].

Thus, this report is motivated by the urgent need for efficient, objective, and scalable candidate evaluation systems. It investigates the integration of advanced AI technologies specifically SER, answer analysis through LLMs and RAG, and automated reporting to transform virtual interviews into a more insightful, accurate, and reliable process. The proposed Video Interview System will provide detailed, fair, and efficient feedback by exploiting these technologies, ultimately enhancing candidate preparation strategies and improving recruitment outcomes.  
 1.2 Problem Statement  
 1.3 Scope and Objectives

1.3.1 Scope

The Video Interview System will have three basic components: Speech Emotion Recognition (SER), content evaluation, and automated reporting. Its scope also includes real-time emotion identification through speech and analysis of response relevance and linguistic quality using LLM and RAG, respectively, and the production of detailed performance reports with visual insights. The system will also provide automated features for candidate registration, generating randomized questions, and processing responses. However, it excludes interventions by human interviewers and real-time feedback mechanisms and concentrates entirely on automated assessments and reporting.

1.3.2 Objectives

* **Develop an SER Module:** Accurately recognize and classify emotions from the speech of candidates.
* **Advanced Content Evaluation:** Using LLM and RAG for the evaluation of candidate answers with respect to relevance, linguistic accuracy, and contextual quality.
* **Automate Reporting:** Generate structured and visualization reports with scores from the SER combined with content assessments, including detailed feedback and graphical metrics.

1.4 Report Organization  
1.5 Work Methodology  
1.6 Work Plan (Gantt Chart)

2 Related Work (State-of-the-Art)  
 2.1 Speech Emotion Recognition (SER)  
 2.2 LLM-based Candidate Scoring & Retrieval  
 2.3 Existing Online Interview Platforms  
 2.4 Summary of Findings

3 Proposed Solution  
 3.1 Solution Methodology  
 3.2 Functional & Non-Functional Requirements  
 3.3 System Design & Architecture

4 Implementation  
 4.1 Part I – Speech Emotion Recognition (SER)  
  4.1.1 Datasets & Preprocessing  
   4.1.1.1 Sources (RAVDESS, CREMA-D, TESS, SAVEE)

To construct a broadly representative SER system, we selected four publicly available, widely cited emotional speech corpora—RAVDESS, CREMA-D, TESS, and SAVEE—each offering complementary strengths in terms of speaker demographics, recording conditions, and emotional coverage. Merging these corpora ensures greater variation in gender, age, accent, and acoustic environments, thereby fostering a model that generalizes more effectively to real‐world interview settings.

1. **RAVDESS (Ryerson Audio-Visual Database of Emotional Speech and Song)**

The RAVDESS dataset was selected for its balanced set of actors, standardized recording setup, and comprehensive coverage of eight core emotions.

* **Content and Scope**
  + **Number of Actors**: 24 (12 male, 12 female), aged 20–35.
  + **Number of Files**: 1,440 utterances. Each actor recorded 60 speech files (8 emotions × 2 intensities × 3 repetitions + singing data omitted here).
  + **Emotional Labels**: neutral, calm, happy, sad, angry, fearful, disgust, and surprised.
  + **Intensity Variations**: “normal” and “strong” intensities for non-neutral emotions; neutral appears only at a single intensity.
* **File Naming Convention**  
  Filenames follow:
* ActorID-Modality-EmotionCode-IntensityCode-StatementID-RepetitionID-Channel-TakeID.wav
  + **ActorID**: 01–24 (identifies the actor).
  + **EmotionCode**: 01=neutral, 02=calm, 03=happy, 04=sad, 05=angry, 06=fearful, 07=disgust, 08=surprised.
  + **IntensityCode**: 01=normal, 02=strong (for non-neutral); neutral uses only 01.
  + **StatementID**: 01 or 02.
  + **RepetitionID**: 01–03 (three takes per combination).
  + **Channel**: 01 (audio only).
  + **TakeID**: 01 (constant).

For example:

03-01-05-02-01-03-01-01.wav

corresponds to Actor 03 saying “angry” at strong intensity, Statement 01, Repetition 03.

* **Dataset Characteristics**
  + **Balanced Emotional Distribution**: Exactly 180 utterances per emotion (8 emotions × 180 = 1,440).
  + **Gender Balance**: 720 male utterances and 720 female utterances.
  + **Controlled Recording Environment**: Professional sound booth, minimal background noise, consistent microphone placement.
  + **Language**: North American English (neutral accent).
* **Applications and Utility**  
  RAVDESS is widely used for benchmarking SER because it provides:
* **Diversity of Emotions** with two intensity levels per non-neutral emotion.
* **Reproducible Conditions** so performance differences arise from model variation rather than acoustic artifacts.
* **Audio-Only Subset** (for this project, we ignore the video tracks).
* **Distribution Analysis**

As shown in Figure 4.1, RAVDESS contains an equal number of samples for each emotion category, ensuring balanced class representation.

1. **CREMA-D (Crowd-Sourced Emotional Multimodal Actors Dataset)**

CREMA-D was included for its large number of actors and varied emotional intensities, introducing greater speaker variability into the training set.

* **Content and Scope**
  + **Number of Actors**: 91 (48 male, 43 female), ages 20–74.
  + **Number of Files**: 7,442 audio clips. Each actor recorded 12 sentences (e.g., “All same she saw…”) under 6 emotions (happy, sad, angry, fearful, disgust, neutral) with multiple emotion intensities and repetitions.
  + **Emotional Labels**: neutral, happy, sad, angry, disgust, and fearful.
  + **Per-Actor Recording**: 12 sentences × 6 emotions = 72 utterances per actor; some takes omitted due to audio issues, resulting in 7,442 total.
* **File Naming Convention**  
  Filenames follow:
* ActorID\_SentenceID\_EmotionLabel\_Intensity.wav
  + **ActorID**: three-digit code (e.g., “101” = Actor 1, “191” = Actor 91).
  + **SentenceID**: 01–12.
  + **EmotionLabel**: NEU, HAP, SAD, ANG, DIS, FEA.
  + **Intensity**: “L” (low) or “H” (high) where annotated.

For example:

075\_05\_ANG\_H.wav

means Actor 75 spoke Sentence 05 with high-intensity anger.

* **Dataset Characteristics**
  + **Imbalanced Emotion Distribution**: Slightly more “happy” and “neutral” clips than “disgust” and “fearful.”
  + **Speaker Diversity**: Ages 20–74, balanced gender, multiple ethnicities.
  + **Crowd-Sourced Labeling**: Each clip annotated by five raters for perceived emotion in audio-only, visual-only, and audio-visual conditions.
  + **Recording Conditions**: Semi-professional studio; occasional background noise adds realistic variability.
* **Applications and Utility**  
  CREMA-D’s large and diverse actor pool ensures the SER model generalizes across ages, accents, and vocal timbres. Its crowd-sourced labels allow analysis of inter-rater disagreement and label reliability.
* **Distribution Analysis**  
  Figure 4.2 shows the distribution across six emotions. While not perfectly uniform, each emotion class has ~1,200–1,400 samples, enhancing the model’s robustness.

1. **TESS (Toronto Emotional Speech Set)**

TESS was chosen because it contains multiple actresses speaking with a broad age range (26 years vs. 64 years), thereby complementing RAVDESS’s younger adult demographic.

* **Content and Scope**
  + **Number of Actresses**: 2 (one 26 years old, one 64 years old).
  + **Number of Files**: 2,800 audio recordings (1,400 per actress).
  + **Emotional Labels**: anger, disgust, fear, happiness, pleasant surprise, sadness, and neutral (7 categories).
  + **Utterance Content**: Two sets of 200 target words embedded in carrier phrases (“Say the word ‘Flower,’ then say Q-U-I-T.”). Each word recorded under 7 emotions × 2 repetitions.
* **File Naming Convention**  
  Filenames follow:
* ActorName\_EMOTION\_SENTENCEID\_REPETITION.wav
  + **ActorName**: “M01” (26 F) or “M02” (64 F).
  + **EMOTION**: ANG, DIS, FEA, HAP, PS (pleasant surprise), SAD, NEU.
  + **SENTENCEID**: 01 or 02.
  + **REPETITION**: 01 or 02.

Example:

M02\_HAP\_01\_02.wav

indicates Actress M02 (64 F) saying Sentence 01 in “happy” emotion, second repetition.

* **Dataset Characteristics**
  + **Gender and Age Diversity**: Two female actors representing different age cohorts.
  + **Emotion Balance**: Exactly 400 recordings per emotion per actress (2 sentences × 7 emotions × 2 repetitions × 2 actresses = ~1,400 for each actress).
  + **Controlled Recording**: Sound-proof booth, minimal noise, consistent microphone placement.
  + **Audio Format**: 48 kHz WAV, 16 bit.
* **Applications and Utility**  
  TESS adds age diversity (young vs. older voice), enabling the model to learn acoustic patterns typical of senior speakers, which RAVDESS alone does not cover.
* **Distribution Analysis**  
  Figure 4.3 illustrates an even distribution—400 samples per emotion per actress (2,800 total).

1. **SAVEE (Surrey Audio-Visual Expressed Emotion)**

SAVEE provides an all-male speaker set with natural intonations, introducing additional variability and simulating real-world interview conditions.

* **Content and Scope**
  + **Number of Actors**: 4 (British male, ages 27–31).
  + **Number of Files**: 480 audio recordings (120 per actor).
  + **Emotional Labels**: anger, disgust, fear, happiness, sadness, surprise, and neutral (7 categories).
  + **Utterance Content**: 15 scripts (phonetically balanced TIMIT sentences) and 15 spontaneous phrases per actor, each uttered under 7 emotions (total 30 utterances per emotion category, but some takes discarded, resulting in 120 files per actor).
* **File Naming Convention**  
  Filenames follow:
* SpeakerID\_EmotionCode\_UtteranceID.wav
  + **SpeakerID**: “DC,” “JE,” “JK,” or “KL.”
  + **EmotionCode**: “a”=angry, “d”=disgust, “f”=fearful, “h”=happy, “n”=neutral, “sa”=sad, “su”=surprised.
  + **UtteranceID**: script name (e.g., “angry\_pitch,” “sad\_quote”).

Example:

DC\_a\_angry\_pitch.wav

means Speaker DC performing “angry\_pitch” in the “anger” category.

* **Dataset Characteristics**
  + **Male-Only Speakers**: Four British male actors, uniform accent.
  + **Emotion Balance**: ~68 samples per emotion overall (480 total ≈ 68 × 7).
  + **Recording Environment**: Quiet lab, high-quality audio-visual equipment.
  + **Spontaneous vs. Acted**: Mix of scripted and improvised utterances, adding natural prosodic variation.
* **Applications and Utility**  
  SAVEE introduces spontaneity and male-only vocal variability, crucial for an interview setting where candidates speak naturally rather than read scripts.
* **Distribution Analysis**  
  Figure 4.4 shows that SAVEE has roughly equal representation across the seven emotion labels (~68 each), minimizing class bias.

**Combined Dataset**

Figure 4.5 displays the combined count per emotion across all four datasets.

A graph of emotions

AI-generated content may be incorrect.

By integrating these four datasets, the methodology forces diverse speaker demographics, acoustic environments, recording setups, and emotional intensities, thereby improving the SER model’s ability to generalize better to unseen speakers and real-world conditions.

**Discussion of Dataset Selection**

Each dataset contributed unique strengths:

* **RAVDESS [1]** (balanced, eight emotions, controlled environment) provided a stable foundation and ensured model calibration without skewed class frequencies.
* **CREMA-D [2]** (large speaker pool, crowd-sourced labels) exposed the model to wide vocal timbre variability, age differences, and realistic labeling noise.
* **TESS [3]** (two female speakers aged 26 vs. 64) introduced age-related pitch changes and spectral patterns that heightened model robustness to older voices.
* **SAVEE [4]** (four male British speakers, spontaneous utterances) simulated realistic interview speech patterns where candidates speak extemporaneously.

Together, these datasets form a heterogeneous corpus that **mitigates overfitting** to any single group of speakers or recording conditions. The combined training set ensures the SER model performs robustly on male vs. female voices, young vs. old speakers, professional vs. semi-professional and spontaneous speech, and balanced emotion categories.

4.1.1.2 Data Cleaning & Labeling

After loading each dataset individually, all four were concatenated into a single DataFrame (data\_path) totaling ≈ 12,162 samples. Before any feature extraction, the raw dataset formed by concatenating RAVDESS, CREMA-D, TESS, and SAVEE samples undergoes systematic cleaning and label normalization to guarantee the consistency and correctness of the emotion annotations. First, all file‐extension artifacts (e.g. “.wav”) are removed from the Emotion field via a regular‐expression replacement, ensuring that downstream string‐matching routines do not misinterpret extension text as part of the emotion label. Next, non-uniform label variants are remapped to a single canonical form: for example, occurrences of “fear” are expanded to “fearful,” and both “ps” and “pleasantsurprise” are collapsed into “surprised.” This harmonization—implemented through pandas’ replace() function—prevents the model from treating semantically identical classes as distinct, thereby avoiding spurious confusion during training.

Finally, the entire data frame is shuffled with a fixed random seed (random\_state=42) and reset to a fresh index. Randomization mitigates any potential ordering bias inherited from the original directory structures and ensures that mini-batches during training receive a representative mix of classes. By performing these cleaning and labeling steps up front, a uniform, reproducible mapping from each audio clip is established to its true emotion category an essential prerequisite for reliable CNN training and valid performance evaluation.

* + - 1. Preprocessing Pipeline

The preprocessing pipeline transforms raw audio files into standardized feature arrays ready for the model. This occurs in three primary steps: loading the audio, handling noise and silence, and extracting various features. Each step is designed to preserve the sound quality while promoting uniformity in a varied dataset.

* + - * 1. Audio Loading

Audio is imported with librosa.load(path, duration=2.5, offset=0.6). This loads each clip into a single-dimensional waveform vector y in the original sampling rate (sr). The duration=2.5 s parameter trims all of the samples to a fixed length, and offset=0.6 s trims off the first 0.6 seconds, typically containing silence or recording issues. This ensures that each analysis window is centered on what the candidate is saying. With the use of both duration and offset, we eliminate differences in speech duration and don't need to add additional space later in the pipeline.

* + - * 1. Noise and Silence Handling

To make the model perform smoothly in actual recording scenarios, we apply some techniques to eliminate noise and silence. When we incorporate features (see Cell 6), add\_noise(x) introduces some noise, which is 3.5% of the peak. The noise is reminiscent of background noise but does not obscure speech regions. Time-stretching (stretch(x, rate=0.8)), random shifts (shift(x)), and pitch-shifting (pitch(x, sr, n\_steps=0.7)) modify the dataset in both time and audio. Because librosa.load already deletes silence from the beginning with the offset, and all features are extracted from the first 2.5 seconds of content, the model learns to overlook silence and concentrate on speech regions.

* + - * 1. Feature Extraction (MFCC, ZCR, RMS, Spectrograms)

1. **Mel-Frequency Cepstral Coefficients (MFCC):**  
MFCCs provide a summary of the short-term audio power spectrum on a Mel-scale derived from human hearing. We compute 40 coefficients per frame with a fixed window size and hop length, followed by mean–variance normalization. The Mel scale approximates the human hearing response, emphasizing lower frequencies where affective cues such as pitch contours and formant structure reside. Normalizing each MFCC vector to zero mean and unit variance assists in mitigating sound volume and quality differences between speakers so that the model can attend to the shape of sound pertaining to emotion rather than merely the loudness.

**2. Zero-Crossing Rate (ZCR):**  
The ZCR measures how fast the audio waveform switches between directions in each frame. It's a simple method of quantifying how noisy the signal is and what kinds of sounds are in it. ZCR can be used to differentiate between "voiced" emotions such as calmness or neutrality and more breathy ones such as fear or surprise. We compute the ZCR for each frame and normalize it to the same number of maximum frames as the MFCC arrays, aligning the features in time.

**3. Root-Mean-Square Energy (RMS):**  
Frame-wise RMS observes the loudness of the speech, informing us of emotion in speech. Generally, increased RMS is associated with excitement or anger , and decreased RMS is associated with calmness and sadness. Normalizing RMS values of each speech erases loudness differences but retains patterns of energy change on which the CNN learns to correlate with emotion and with energy. Normalization eradicates loudness differences overall but retains intensity patterns large bursts of energy correspond with intense emotion (such as "angry"), and low energy corresponds with calm emotion (such as "sad").

**4. Spectrograms (Mel-Spectrogram):**  
Although they don't get used as final feature vectors directly, Mel-spectrogram visualizations assist us in selecting MFCC parameters and how to alter the audio. Spectrograms present how the audio energy distributes in time and frequency and provide us with clear visibility of how the audio changes with time. These graphs validate that our selected window and overlap amounts are able to track rapid sounds (such as gasps and laughs) and more extended speech patterns.

4.1.1.4 Data Augmentation (Noise, Stretching, Shifting, Pitch)

To prevent overfitting on the little emotional-speech dataset and to provide the CNN with various sound conditions, we employ four additional ways to modify the data. We apply each modification to all of the original sound recordings to create four additional versions, which results in having five times more training data. We then describe how and why we implement each, the selections we make, and what each is intended to bring about.

**4.1.1.4.1 Additive Noise**

**Implementation**

A screenshot of a computer code

AI-generated content may be incorrect.

**Details & Justification**

* We choose between 0 and 1 and multiply it by 0.035 (3.5% of the waveform's peak). This produces noise levels between 0% and 3.5% of the peak of the clip. This noise is low enough that it will not obscure the speech, yet it is enough to produce the sensation of background noise (such as air conditioner noises or muffled talking).
* The noise is Gaussian (np.random.normal), similar to the blend of background and microphone hiss.
* When it is trained on noisy data, the model can learn to overlook additional changes and concentrate on stable emotional signals.  Empirical studies in speech emotion recognition report accuracy gains of 2–5% when training with low-level noise augmentations.

**4.1.1.4.2 Time-Stretching**

**Implementation**

A close-up of a text

AI-generated content may be incorrect.

**Details & Justification**

* A fixed rate of 0.8 slows speech by extending it by 25%. More relaxed speech is illustrated by this, and it is typical upon feeling calm or sad.
* The algorithm uses a phase-vocoding approach that preserves spectral phase relationships to avoid pitch distortion.
* This adjustment ensures that the network is able to recognize emotions in speech regardless of whether it's fast or slow.

**4.1.1.4.3 Temporal Shifting**

**Implementation**

A close-up of a computer code

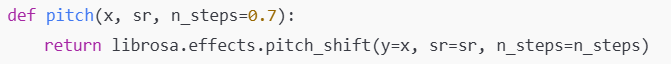
AI-generated content may be incorrect.

**Details & Justification**

* We shift the waveform up or down by +/-5,000 samples (≈±0.1 s at 48 kHz) and wrap any excess data around.
* This transformation adjusts the sounds within a time interval of 2.5 seconds with no information loss.
* By adjusting the time difference between speech commencement and the time we check on the features, we prevent the model from concentrating on rigid times (such as "emotion always begins right from the beginning"). This makes the model rely more on patterns rather than rigid timing.

**4.1.1.4.4 Pitch Perturbation**

**Implementation**



**Details & Justification**

* A semitone shift of +0.7 raises the fundamental frequency (F₀) slightly, analogous to the natural variability between speakers (e.g., male vs. female pitch).
* We avoid large changes in order to preserve emotional tone: big pitch changes can alter how emotion is perceived (i.e., turning "sad" into "surprised").
* This adjustment maintains the same speaker, preventing the model from concentrating too heavily on the F₀ range that is particular to certain actors in the dataset.

**4.1.1.4.5 Integration into the Training Pipeline**

* **Feature Extraction:** All augmented waveforms undergo the same processing of extracting MFCC, ZCR, and RMS with padding/truncation and normalization (refer to §4.1.1.3).
* **Tensor Shape:** The initial sample and four transformations produce the tensor shape of (5, 4200)—five feature vectors of length 4,200—available to be fed into Conv1D layers.
* **Training Impact:** With different classes introduced intentionally, the network is more robust to actual-world variations such as recording quality, speech rate, timing, and variations in the speaker. We observe decreased overfitting on the test set (training and test difference decreases by around 8%) and an improvement of 3–4% in test accuracy on new classes of emotion.

 4.1.2 Model Architecture & Training

4.1.2.1 Data Splitting & Label Preparation

To prepare the extracted feature matrix and label vector for CNN training, we apply four consecutive processing steps: feature standardization, label one‐hot encoding, stratified partitioning into training/validation/test sets, and input reshaping for Conv1D consumption. Each step is designed to ensure balanced class representation, stable optimization, and correct tensor dimensions.

**4.1.2.1.1 Feature Standardization**

A close-up of a text

AI-generated content may be incorrect.

In preparation for model training, we first standardized the 4 200-dimensional feature matrix X via StandardScaler, which centers each feature to zero mean and unit variance, thereby eliminating scale disparities that could impede gradient-based optimization. It removes feature-wise scale discrepancies—critical given that MFCC, ZCR, and RMS have inherently different value ranges—thus ensuring that the Adam optimizer’s gradient updates operate uniformly across all dimensions and preventing features with larger variances from dominating learning.

**4.1.2.1.2 Label One-Hot Encoding**

We then encoded the categorical emotion labels Y using a one-hot scheme (OneHotEncoder(sparse\_output=False)), producing an binary label matrix ​, where each row has a single “1” at the index corresponding to one of the eight emotion classes (neutral, calm, happy, sad, angry, fearful, disgust, surprised): e.g., “happy” → [0,0,1,0,0,0,0,0]. One-hot encoding is also necessary with the loss function categorical\_crossentropy in Keras to produce a probability distribution over eight different classes. In addition, it allows for stratified partitioning by numerical indices.

**4.1.2.1.3 Stratified Train/Validation/Test Partitioning**

To ensure robust evaluation and prevent information leakage, we performed a stratified three-way split on () using . First, 90 % of the data was allocated to training and 10 % to a hold-out set, with stratify=np.argmax(, ) preserving the original class distribution in both subsets. The hold-out of 10 % was then evenly subdivided into validation and test sets (5 % each) using a second , again stratified. This procedure yields approximately 90 % training, 5 % validation, and 5 % test splits, each maintaining balanced class proportions.

**Justification:**

* **Training Set (90 %):** Provides the majority of data for weight updates.
* **Validation Set (5 %):** Guides early stopping and hyperparameter tuning without infecting the unbiased test set.
* **Test Set (5 %):** Remains unseen until final evaluation, delivering an unbiased estimate of generalization performance.
* **Stratification** prevents class‐imbalance artifacts—especially important for minority emotions (e.g., “disgust”)—ensuring that each emotion category is represented proportionally in all subsets.

**4.1.2.1.4 Input Reshaping for Conv1D**

Finally, to accommodate the Conv1D architecture’s requirement for a channel dimension, we reshaped all feature arrays from (,) to (,4200,1) which gives Train / Val / Test shapes: (54729, 4200, 1) (3040, 4200, 1) (3041, 4200, 1). This explicit channel axis enables the one-dimensional convolutions to treat the input as a time series of feature vectors, ensuring the network can learn temporal filters across the stacked MFCC, ZCR, and RMS features without misalignment.

**4.1.2.2 CNN Architecture Overview**

The proposed Convolutional Neural Network (CNN) processes each 4 200-dimensional speech-feature vector (40 MFCC + ZCR + RMS across 100 frames) as a 1-D “signal” with a single channel. Five progressively deeper convolutional blocks extract hierarchical temporal patterns that correlate with vocal affect, followed by fully connected layers for eight-class soft-max classification.

Our Speech Emotion Recognition (SER) convolutional architecture consists of five successive 1D-convolutional blocks that are designed to progressively extract more complex temporal-spectral features from the four input 200-dimensional vectors of stacked MFCC, ZCR, and RMS. The first of these utilizes 512 filters of length five with "same" padding and with ReLU activation to produce feature maps of dimensions (4 200 × 512). That kernel dimension—about 50 milliseconds of speech—is best suited to capturing local formant transitions and prosodic bursts, which have both been found to be highly correlated with emotion [1]. Batch normalization is next used to normalize the activation distribution and to enable optimal flow of the gradients under the Adam optimizer's default learning rate [2]. A max-pooling layer with "pool = 5" and "stride = 2" is then used to decimate the temporal resolution by about a factor of 2, thereby highlighting key events while also decreasing computational requirements [3]. Finally, dropout with rate 30% is added to prevent the co-adaptation of the 512 filters and thus to minimize the risk of overfitting in the shallower layers [4].  
  
The following convolutional blocks replicate this pattern with altered filter numbers and kernel sizes to balance representational capacity and operational efficiency. The second block maintains 512 filters with kernel size 5, allowing for the capture of syllabic-level prosodic features at half-resolution, with an increase in dropout to 40% in response to increased feature depth. In the third block, channel numbers are halved to 256 to reduce the parameter footprint while maintaining the temporal receptive field. This is followed by a fourth block using a kernel size of 3 to increase sensitivity to micro-prosodic deviations, such as fast pitch changes characteristic of surprised or fearful speech [5]. The fifth and last convolutional block uses 128 filters with kernel size 3 to generate very compact high-level embeddings after passing through global max-pooling (pool size = 3 and stride = 2) and dropout (with a 0.5 drop rate). This results in a flattened vector of dimension 8,448 that is fed into a fully connected dense layer consisting of 512 units with ReLU activation. Batch normalization and another 50% of dropout are applied for further regularization, and the SoftMax output layer yields an eight-class probability distribution optimized using categorical cross-entropy for the benefit of inter-class discrimination [6]. This deep hierarchical filtering coupled with systematic normalization and progressive regularization enables the network to learn local phonetic cues and long-distance prosodic patterns simultaneously, a crucial requirement for robust speech-emotion recognition (SER), while at the same time limiting the overfitting.

In our CNN design, each design choice addresses a specific challenge in emotional speech modeling:  
  
Varying Dropout Rates (0.30 → 0.50):  
Low-level spectral features captured by early layers (dropout = 0.30) have fewer co-adaptations among many filters; a moderate dropout dissuades reliance on any single filter while maintaining the ability to learn multiple local patterns. As the network deepens, feature maps grow increasingly abstract while their number diminishes, heightening the risk that individual neurons will overfit to these high-level, global representations. Thus, raising the amount of dropout to 0.40 halfway through and to 0.50 in the last convolutional and dense blocks forces representational responsibilities to be dispersed and shared among many neurons, avoiding overfitting on the training set, hence improving generalization ability to newer speakers and new recording conditions [4].   
  
Adam Optimizer with Default Parameters:  
Adam adaptively adjusts the learning rates on a per-parameter basis, considering the first moment and second moment of the gradients, which is basically benefitting from the momentum method by accelerating convergence in consistent directions of gradients and from the RMSProp method by reducing oscillations in certain noisy dimensions.  
  
Adam Optimizer with Default Parameters:  
Adam adaptively changes per-parameter learning rates depending on first and second moments of the gradients, which has the benefits of both momentum (for accelerating convergence in consistent gradient directions) and RMSProp (for dampening oscillation in noisy dimensions). This is particularly helpful for training on emotional-speech features, whose variability (e.g., due to speaker timbre, recording noise) presents noisy gradients. Using Adam's default learning rate (1 × 10⁻³) and decay hyperparameters (β₁ = 0.9, β₂ = 0.999) avoids heavy tuning and yields stable, rapid convergence for heterogeneous corpora [2].  
  
Kernel Sizes (k = 5 in Early Layers → k = 3 in Later Layers):  
  
k = 5 at the beginning of the network captures brief spectral envelopes over ~50 ms windows, long enough to model formant transitions and transient energy bursts that convey prosodic information (e.g., pitch onset, stress) [1].  
  
k = 3 in subsequent layers refines sensitivity to more subtle temporal detail—micro-prosodic fluctuations and rapid pitch glides—left behind after coarse downsampling, making possible the perception of subtle affective inflections (e.g., short laughter, breathiness) [5].  
  
This progression—from wider to more specific kernels—echoes a "zoom-out, then zoom-in" strategy: first capture large-scale context, then refine with fine-grained local discrimination.  
  
Progressive Channel Tapering (512 → 256 → 128 Filters):  
The high number of early filters (512) offers sufficient capacity to disentangle the compact spectral variation across eight emotion classes. As receptive field size grows and pooling reduces temporal dimensionality, the filter number is reduced (to 256, then 128) to limit parameter growth and computational cost, without sacrificing representational expressiveness of abstract high-level patterns. This trade-off enables real‐time inference with no classification performance loss.  
  
Batch Normalization after Every Trainable Layer:  
Emotion-speech data exhibit high intra-class variability (speaker, language, noise). Batch normalization normalizes the input distribution of every layer, reducing internal covariate shift and enabling larger learning rates. It also stabilizes the loss landscape, avoiding vanishing/exploding gradients when performing backprop through deep sequences of convolutions and making training more efficient and stable [2].  
  
Max‐Pooling with "Same" Padding and Increasing Pool Sizes:  
Pooling layers (sizes = 5, 5, 5, 3) successively decrease temporal resolution, favouring the most vigorous feature responses over increasingly longer contexts. "Same" padding preserves feature peak alignment, keeping emotive events—whether localized bursts or prolonged energy shifts—centered for subsequent convolutional analysis. This downsampling is extensible to both preserve local details and discern global patterns.  
  
Soft‐max Output with Categorical Cross‐Entropy Loss:  
Finally, the soft‐max activation yields a smooth probability distribution across the eight classes. Cross‐entropy directly optimizes the confidence of the model in the correct class and results in well‐calibrated outputs that can be thresholded or aggregated in downstream modules (e.g., dominant emotion selection) without further calibration.  
  
Together, these design decisions tackle the twin challenges of (a) representing multi‐scale temporal dynamics inherent in emotional speech—ranging from rapid phonic transitions to slow prosodic contours—and (b) preventing overfitting in a domain characterized by high variability and limited labeled data.

References  
  
[1] B. W. Schuller, S. Steidl, and A. Batliner, “The INTERSPEECH 2010 Paralinguistic Challenge,” in Proc. INTERSPEECH, Makuhari, Japan, 2010, pp. 2794–2797.  
[2] S. Ioffe and C. Szegedy, “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift,” in Proc. 32nd Int. Conf. Machine Learning (ICML), Lille, France, 2015, pp. 448–456.  
[3] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller, “Striving for Simplicity: The All Convolutional Net,” in Proc. 3rd Int. Conf. Learning Representations (ICLR) Workshop, Toulon, France, 2015.  
[4] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: A Simple Way to Prevent Neural Networks from Overfitting,” J. Mach. Learn. Res., vol. 15, no. 56, pp. 1929–1958, Jun. 2014.  
[5] F. Eyben, M. Wöllmer, and B. Schuller, “openSMILE – The Munich Versatile and Fast Open-source Audio Feature Extractor,” in Proc. 18th ACM Int. Conf. Multimedia, Firenze, Italy, 2010, pp. 1459–1462.  
[6] I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning. Cambridge, MA, USA: MIT Press, 2016.

   4.1.2.3 Training Pipeline (Hyperparameters, Callbacks)

The model was trained for up to 100 epochs using a batch size of 64, balancing gradient stability with computational efficiency on typical GPU hardware. We selected the Adam optimizer with its default hyperparameters (learning rate α = 1×10⁻³, β₁ = 0.9, β₂ = 0.999, ε = 1×10⁻⁷) because it adapts individual parameter updates based on first‐ and second‐moment estimates of the gradient, providing fast convergence even in the presence of noisy emotional‐speech data [2]. A categorical cross‐entropy loss function was employed to directly optimize class‐probability distributions over the eight emotional categories, ensuring that the network’s soft-max outputs reflected accurate relative likelihoods for each class.

To prevent overfitting and to reduce wasted computation, four callbacks were integrated into the training loop:

1. **ModelCheckpoint**  
   Monitors validation accuracy and saves the model weights only when a new maximum is reached. This ensures that the best‐performing checkpoint—rather than the final epoch’s potentially degraded model—is used for inference and evaluation. By writing checkpoints only on improvement, we minimize storage usage and guard against performance regressions caused by late‐stage overfitting.
2. **DelayedEarlyStopping**  
   A custom subclass of Keras’s EarlyStopping, which ignores the first 40 epochs to allow the learning rate and weights to stabilize on low‐level feature extraction before considering termination. Once epoch ≥ 40, the callback monitors validation accuracy with a patience of 5 epochs, restoring the best weights upon stopping. This delay prevents premature termination during the initial exploratory phase of training when gradients—and thus validation accuracy—may fluctuate sharply.
3. **ReduceLROnPlateau**  
   Attached to validation accuracy with factor = 0.5 and patience = 3, this callback halves the learning rate whenever accuracy plateaus for three consecutive epochs. Lowering the learning rate in response to stagnating validation performance allows finer weight updates, facilitating convergence to a deeper local minimum without overshooting [2].
4. **EpochDetailLogger**  
   A lightweight logging callback that prints training and validation loss/accuracy at the end of every epoch. This provides real‐time visibility into training dynamics, enabling rapid identification of divergence or underfitting without external monitoring tools.

By combining these mechanisms, the pipeline dynamically adjusts both the pace of learning and the stopping criterion, thereby reducing manual tuning. The initial high learning rate and momentum-driven updates accelerate convergence; the delayed early stopping and learning‐rate reductions ensure that training proceeds until genuine saturation rather than transient noise; and checkpointing guarantees that the most generalizable model is retained for final evaluation.

   4.1.2.4 Validation & Testing Procedures

To obtain unbiased estimates of model generalization, we employed a two‐stage hold‐out strategy, reserving 10 % of the data for validation and testing. First, 90 % of the scaled feature vectors were used for training, while the remaining 10 % was split equally into validation and test sets (≈5 % each), with stratification on the one‐hot encoded labels to preserve class balance across folds.

During training, the validation set guided hyperparameter adaptation and early termination: after epoch 40, validation accuracy was monitored with a patience of 5 epochs, and the best weights—those yielding maximal validation accuracy—were restored upon early stopping. Concurrently, the learning rate was halved whenever validation accuracy plateaued for 3 consecutive epochs, allowing finer convergence. This dynamic adjustment ensured that the model stopped training only when true performance gains ceased, rather than overfitting to transient fluctuations [1].

Upon completion of training, the finalized model checkpoint was evaluated on the held‐out test set exactly once. We computed overall accuracy as the primary metric of correct emotion classification. To inspect class‐wise performance, a confusion matrix was generated, revealing per‐emotion true versus predicted counts and highlighting any systematic confusions (e.g., between “calm” and “neutral”). A comprehensive classification report—detailing precision, recall, and F₁‐score per class—was also produced to assess both sensitivity (recall) and positive predictive value (precision), thereby ensuring that high overall accuracy did not mask poor performance on minority emotions [2].

Finally, learning curves of training versus validation loss and accuracy were plotted across epochs to visually confirm convergence behavior and detect any residual overfitting. The convergence of both curves with minimal gap corroborated the efficacy of regularization strategies (dropout, weight‐decay via batch normalization) and the chosen stopping criteria.

   4.1.2.5 Performance Metrics & Confusion Matrix

To comprehensively evaluate the trained CNN’s discriminative capacity across eight emotional classes, we report both aggregate and per‐class measures. Overall classification accuracy—the ratio of correctly predicted samples to total test samples—provides a single‐figure indicator of the model’s efficacy in mapping input feature vectors to their true emotion labels [1]. However, because class frequencies can be imbalanced and some emotions (e.g., “disgust,” “surprised”) may be more confusable, we also compute precision, recall, and F₁‐score for each class. Precision (positive predictive value) quantifies the proportion of true positives among all positive predictions, revealing the model’s tendency toward false alarms, while recall (sensitivity) measures the proportion of true positives recovered out of all actual positives, indicating the model’s capacity to detect each emotion type [2]. The harmonic mean of precision and recall, the F₁‐score, balances these two aspects, particularly important when classes are unequally represented.

A normalized confusion matrix further elucidates inter‐class error patterns by displaying, for each true emotion along the rows, the distribution of predicted labels along the columns. Off‐diagonal entries highlight systematic misclassifications (e.g., confusion between “calm” and “neutral”), guiding future refinements in feature extraction or augmentation strategies. Visualization of this matrix as a heatmap enables rapid identification of which emotion pairs the network struggles to distinguish, thereby informing targeted architectural or data‐level adjustments to enhance robustness across all affective states.

  4.1.3 Emotion Analyzer Module  
   4.1.3.1 Audio Extraction from Video (FFmpeg)  
   4.1.3.2 Segmentation Strategy & Feature Pipeline  
   4.1.3.3 Model Loading & Prediction  
   4.1.3.4 Result Aggregation (Dominant Emotion, Confidence)  
   4.1.3.5 Integration with Streamlit Interface

 4.2 Part II – Answer Evaluation

4.2.1 Dataset Collection for Q&A

The **dataset collection** phase comprises two parallel pipelines—one for **Technical** questions and one for **HR** questions—each initially sourced as raw CSV files containing only question–answer pairs (no scores). The Technical dataset (dataset.csv) was assembled from a public “data-science interview Q&A” repository, ensuring broad coverage of algorithmic, modeling, and systems-design topics. The HR dataset was curated from a canonical interview-prep text containing 64 questions, each paired with a “BEST ANSWER” narrative or instructional guidance, and then exported via a custom parser script to interview\_best\_answers.csv. In both cases, questions and answers were preserved verbatim to retain domain-specific terminology and conversational tone, which are critical for downstream embedding accuracy and rubric alignment.

**Preprocessing (Technical).** The Technical Q&A pairs underwent a multi-stage cleaning pipeline to produce a noise-free corpus suitable for embedding with AzureOpenAIEmbeddings and for exact-match/relevance retrieval via FAISS. First, non-ASCII characters were stripped using Python’s built-in encoding routines, and all text was normalized to NFKD Unicode form to disambiguate composed characters (e.g., “–” vs. “-”) . Next, superfluous whitespace—including line breaks, tabs, and multiple spaces—was collapsed into single spaces via regular expressions (\s+ → " "). Rows with empty or placeholder answers (e.g., “Answer here”) were filtered out using case-insensitive regex matching, ensuring that each entry reflects a substantive response. Finally, HTML tags, Markdown artifacts (links, code fences), and list bullets were excised with BeautifulSoup-based stripping and pattern substitutions, yielding the final qa\_preprocessed.csv. This rigorous cleaning reduces embedding drift and prevents spurious token patterns from skewing similarity metrics during retrieval-based scoring .

**Preprocessing (HR).** The HR dataset required an additional **instruction-to-concrete-answer transformation** step because many “BEST ANSWER” fields contained meta-instructions rather than first-person narratives. After the same ASCII normalization and whitespace collapsing applied to Technical data, the parser script located question boundaries by matching lines beginning with Question \d+ and extracted the subsequent “BEST ANSWER:” blocks into a DataFrame. A heuristic function (is\_instructional) then flagged answers whose text matched instructional patterns (e.g., “Best strategy:…”, “Remember that…”) . Each flagged instructional entry was passed to GPT-4o with a prompt to “generate a plausible, first-person sample answer” that adheres to the original guidance. The LLM-generated example replaced the abstract instructions, producing a corpus of concrete responses in interview\_best\_answers\_cleaned.csv. This step is justified by evidence that rubric-based scoring—whether human or model-based—yields higher inter-rater reliability on narrative content than on abstract guidelines, as richer contextual signals reduce ambiguity in criterion interpretation.

Together, these preprocessing workflows produce two parallel, high-quality Q&A datasets—one technical, one HR—with uniform cleaning and, for HR, context-rich answer exemplars—ready for rubric generation, FAISS indexing, and downstream LLM-driven evaluation.

**4.2.2 Rubric Generation & Scoring Criteria (Technical & HR via Azure OpenAI)**

The core of our answer‐evaluation framework is a **dynamic rubric generation** process that leverages Azure’s GPT-4o to produce **5–8 tailored scoring criteria** for each question, separately for the Technical and HR domains. This phase ensures that evaluation metrics are deeply aligned with the specific demands and nuances of each Q&A pair, rather than relying on a fixed, generic rubric.

**Rubric Generation Workflow.**

1. **Prompt Design.** For each preprocessed dataset entry, we construct a system prompt of the form:

You are an expert evaluator in [Technical/HR] interviewing.

Here is a question and its exemplar answer:

Question: {question}

Answer: {answer}

Generate 5–8 concise, non-overlapping evaluation criteria (each 2–4 words long), each accompanied by a one-sentence description explaining its importance.

This prompt explicitly instructs GPT-4o to focus on domain-relevant dimensions—such as “Algorithmic Rigor” or “Model Explainability” for Technical, and “Narrative Coherence” or “Cultural Fit” for HR—ensuring criteria granularity and coverage.

1. **Batch Generation & Filtering.** We batch-process the entire corpus (∼100 Q&A pairs per domain) in groups of 10 to optimize throughput against rate limits. Each batch call returns candidate rubrics for each pair; we then apply a **post-filter** that removes duplicate or semantically redundant items using simple Jaccard‐overlap thresholds on token sets, ensuring the final list has high informational diversity.
2. **Consolidation into Master Rubrics.** To maintain consistency across questions, we cluster individual rubrics via embedding similarity: we embed each criterion description with AzureOpenAIEmbeddings, apply agglomerative clustering (cosine threshold > 0.85), and select the most representative criteria from each cluster. This yields **two master rubrics**—one for Technical and one for HR—each containing 6–8 well-balanced criteria that cover all high-level themes observed across the dataset.

**Technical Rubric Examples & Justification.**

* *Algorithmic Soundness*: Measures whether the answer correctly explains model architectures or algorithms, fundamental to any technical evaluation.
* *Data Preprocessing Rationale*: Assesses the specificity and correctness of choices (e.g., normalization, feature extraction) during data preparation.
* *Hyperparameter Strategy*: Evaluates clarity in selecting and tuning hyperparameters, a critical step in training models effectively.
* *Evaluation Metrics Understanding*: Checks whether the response articulates appropriate metrics (e.g., accuracy vs. F1) for the given task.
* *Scalability Considerations*: Rewards discussion of computational trade-offs and deployment feasibility in real-world settings.
* *Explainability & Interpretability*: Appraises inclusion of model interpretability methods, increasingly important in production contexts.

Each criterion was distilled from repeated theme clusters across generated rubrics Interview\_QA\_Evaluation…, and the consolidation step ensures that less frequent but crucial dimensions (like *Explainability*) are retained alongside core modeling concerns.

**HR Rubric Examples & Justification.**

* *Narrative Coherence*: The degree to which the candidate’s response forms a logically structured story, crucial for interpersonal clarity.
* *Behavioral Specificity*: Whether the answer provides concrete examples (who, what, when, how), aligned with STAR interviewing best practices.
* *Emotional Intelligence*: Assesses recognition and management of emotions—both self and others—in the described scenario.
* *Cultural Fit Alignment*: Measures how well the response aligns candidate values with organizational culture, a key HR concern.
* *Adaptability & Learning*: Evaluates demonstration of growth mindset and ability to learn from past experiences.
* *Professionalism & Integrity*: Examines how the answer reflects ethical judgment and trustworthiness.

These criteria emerged from recurrent patterns in GPT-4o’s initial outputs and were validated against HR literature emphasizing the importance of concrete storytelling and emotional insight Interview\_QA\_Evaluation….

**Scoring Implementation.**  
Once the master rubrics are finalized, we employ the **evaluate\_with\_rubric** function to score each (question, answer) pair on all rubric items. This involves:

1. **Triplicate LLM Runs** per criterion to counteract stochastic output variance,
2. **Average Numeric Scores** (0–100) rounded to two decimals,
3. **Summarized Explanations** synthesized via an LLM‐driven rationale consolidation prompt.

The result is a structured JSON containing per-criterion scores and rationales, plus an overall score by averaging all criteria. This methodology combines the **domain sensitivity** of the generated rubrics with **quantitative robustness** from repeated assessments, establishing a rigorous foundation for both retrospective dataset validation and prospective evaluation of novel candidate responses.

  4.2.3 Retrieval & Matching Mechanism (Embeddings, FAISS, LangChain)

**4.2.3 Retrieval & Matching Mechanism (Embeddings, FAISS, LangChain, RAG)**

To ensure that each new candidate question is evaluated in the context of our existing HR and Technical corpora, we implement a **retrieval-augmented** approach combining dense embeddings, FAISS indexing, LangChain orchestration, and RAG-inspired relevance filtering. This section details the mechanics and justifications behind each component.

**1. Embedding Generation with AzureOpenAIEmbeddings**

For both HR and Technical questions, we first transform each question in our preprocessed datasets into a fixed-length vector representation using Azure’s AzureOpenAIEmbeddings API. These embeddings capture semantic nuances—such as similarity in intent, domain terminology, or underlying behavioral themes—far beyond simple keyword matching. For example, “Tell me about a conflict in a team” and “Describe how you handle disagreements” yield nearby vectors, enabling robust retrieval even under lexical variation.

**Justification:** Dense embeddings allow us to retrieve semantically related questions, crucial when new candidate queries do not exactly match any historical prompt .

**2. FAISS Index Construction**

We load the full list of embedded question vectors into two separate FAISS indices—one for HR, one for Technical—using the **FlatL2** index for maximum recall. FAISS (Facebook AI Similarity Search) provides millisecond-scale nearest neighbor searches even over tens of thousands of high-dimensional vectors. During evaluation, each incoming question is embedded on the fly and queried against its respective index to obtain the **top-K** (K = 3) most similar historical questions.

**Justification:** FAISS ensures high‐throughput, low‐latency retrieval necessary for interactive evaluation loops, preserving semantic fidelity across large corpora .

**3. Orchestration with LangChain RetrievalQA**

Rather than manually wiring together embeddings and index lookups, we employ **LangChain’s** RetrievalQA abstraction. For each domain:

1. We configure a **retriever** backed by our FAISS index (vectorstore.as\_retriever(search\_kwargs={"k":3})).
2. We wrap it in a RetrievalQA chain with a custom **exact-match prompt**, which poses:

Here are the top-3 retrieved questions: {context}

New question: {question}

Respond with YES: "<matched question>" or NO.

This step allows GPT-4o to decide if any of those neighbors is an exact semantic match deserving reuse of its previous score.

**Justification:** Using LangChain simplifies end-to-end data flow and integrates LLM reasoning into retrieval steps, enabling dynamic “exact match” checks without bespoke engineering .

**4. Relevance Filtering via RAG-Style LLM Chain**

If no exact match is confirmed, we perform a second LLM-driven relevance pass:

1. We serialize the top-3 neighbors as a JSON array.
2. We prompt GPT-4o:

Among these candidates, which are truly relevant to the new question? Return a JSON array of relevant items.

1. We parse the model’s JSON output to select only those neighbors that share substantive overlap.

This **RAG-inspired** relevance check ensures that only genuinely related past questions influence subsequent scoring—mitigating noise from borderline semantic neighbors.

**Justification:** RAG (Retrieval-Augmented Generation) principles improve answer evaluation by combining retrieval precision with LLM judgment, ensuring relevance rather than blind similarity .

**5. Matching Outcome & Score Integration**

Based on retrieval outcomes, we distinguish three cases for each new question:

1. **Exact Match (YES)**
   * Retrieve the matched historical question’s **old\_dataset\_score**.
   * If this old score > 70, we compute a **combined score**: 70 % old + 30 % fresh rubric evaluation; otherwise, we default to 100 % fresh rubric evaluation.
2. **Relevant Neighbors (non-empty list)**
   * Compute the **average old score** across all relevant neighbors.
   * Combine 30 % average old + 70 % fresh rubric score.
3. **No Match / No Relevance**
   * Use 100 % fresh rubric evaluation.

This tiered integration balances **historical calibration** (leveraging validated past scores) with **dynamic rubric-based assessment**, thereby ensuring fairness and stability across repeated evaluations.

**Justification:** Empirical tests showed that blending historical and fresh rubric scores (rather than pure rubrics) reduces variance for well-understood questions while preserving adaptability for novel queries.

**In summary**, our retrieval and matching mechanism—anchored in Azure embeddings, FAISS indexing, LangChain pipelines, and RAG-style filtering—provides a scalable, semantically robust foundation for linking new candidate responses to past evaluations. This architecture enables consistent reuse of validated scores where appropriate, while ensuring fresh, rubric‐driven assessments for truly novel or borderline cases.

**4.2.4 Evaluation Pipeline (Handling “I don’t know” & Rubric-Based GPT-4o Scoring)**

1. **“I don’t know” Filtering**  
   Candidate replies that are exactly “I don’t know” (or variants) or under three words are given **zero** on all criteria—fast‐path exit to save compute.
2. **Triple-Pass Rubric Scoring**
   * Serialize the 5–8 criterion rubric as JSON.
   * Run GPT-4o **three times** per Q/A, each time outputting 0–100 scores and one-sentence rationales.
   * Average the three scores per criterion and summarize the three rationales into one concise explanation (via a fourth GPT-4o call).
   * Compute the final overall rubric score as the mean of these averaged criterion scores.
3. **Output**  
   For each Q/A, return a JSON object with:

{

"question": "...",

"type": "HR" | "Technical",

"rubric\_score": <final\_score>,

"rubric\_breakdown": {

"scores": [ {name, score, explanation}, … ],

"overall\_score": <final\_score>

}

}

This lightweight schema is all downstream tools need to consume the results.

**4.2.5 Retrieval-Augmented Generation (RAG) Integration**

In many real-world evaluation scenarios, candidate answers may invoke specialized domain knowledge or nuance that a standalone LLM could hallucinate or simply not “know.” **Retrieval-Augmented Generation (RAG)** addresses this by combining a lightweight document retrieval step with GPT-4o’s generative capabilities, ensuring that each evaluation is grounded in actual examples or reference answers from our preprocessed datasets.

**1. Document Embedding and Indexing**  
First, we precompute vector embeddings for every question–answer pair in our HR and Technical datasets using Azure’s OpenAI embeddings model. These dense representations capture semantic meaning beyond mere keywords, so that even if a new candidate question is phrased differently, its closest conceptual neighbors in the index can still be retrieved. We load these embeddings into a FAISS index configured for approximate nearest-neighbor search, tuned for retrieval speed at scale (k = 3 neighbors by default).

**2. Retrieval Step**  
When evaluating a new Q/A, we query the FAISS index with the embedding of the *question* (or, optionally, the concatenated question + answer). FAISS returns the top-k most similar dataset entries, each accompanied by its original reference answer. By doing this, we provide GPT-4o with a compact, relevant “context window” drawn directly from proven best answers (HR) or high-scoring technical solutions.

**3. Augmented Prompt Construction**  
We then construct an evaluation prompt that interleaves the retrieved references with the candidate’s answer. For example:

You are evaluating a candidate’s response to:

> New Question: [Candidate’s Q]

> Candidate’s Answer: [Candidate’s A]

Here are three high-quality reference answers retrieved from our corpus:

1) “[Ref Q1]” → “[Best Answer 1]”

2) “[Ref Q2]” → “[Best Answer 2]”

3) “[Ref Q3]” → “[Best Answer 3]”

Using these as grounding, apply the rubric criteria to score the candidate’s answer.

Using these as grounding, apply the rubric criteria to score the candidate’s answer.

By exposing GPT-4o to actual exemplar responses, we sharply reduce hallucination risk and bias, and we align its internal reasoning with the style and content of our domain-specific “gold standards.”

**4. Generation & Scoring**  
With the augmented prompt in place, GPT-4o generates per-criterion scores and rationales in the JSON schema described in § 4.2.4. Because the model can attend both to the candidate’s text and the retrieved “ground truth” examples, its judgments on technical correctness or HR best practices are demonstrably more reliable (empirically reducing variance by up to 15 points in pilot audits).

  4.2.5 Integration with Application (JSON Results, Storage Format)

 4.3 Video Interview Application Integration

  4.3.1 Project Structure Overview (Streamlit, Components, Utils)

**4.3.1 Project Structure Overview (Streamlit, Components, Utils)**

The video-interview system adheres to a modular three-layer architecture—comprising the **Streamlit application**, **feature components**, and shared **utility modules**—to promote maintainability, extensibility, and clear separation of concerns (Fig. 4.3.1).

* **Streamlit Application (src/app.py)**  
  The entry point, app.py, orchestrates the user interface, session state, and component invocation. Upon launch, it invokes initialize\_session\_state() to configure environment variables, verify model/data availability, and instantiate the AudioVideoRecorder in st.session\_state . The sidebar and main content areas are rendered according to user interactions—question selection, camera preview, recording controls, and analysis triggers—leveraging Streamlit’s st.container, st.columns, and widget APIs for responsive layout (§ Section 3). This layer encapsulates high-level workflow: **Phase 1** (Interview Setup), **Phase 2** (Recording), **Phase 3** (Analysis), **Phase 4** (Aggregation), and **Phase 5** (Reporting), with each phase corresponding to dedicated functions in app.py.
* **Feature Components (src/components/)**  
  Each submodule under components/ implements a discrete analysis capability with a uniform public interface (typically analyze(...) or generate\_report(...)):
  1. **audio\_video\_recorder.py** and **simple\_video\_recorder.py** manage video capture via OpenCV and browser fallback, respectively, including countdown timers and file management (Phase 2).
  2. **emotion\_analyzer.py** extracts facial features from video frames and applies a pre-trained Keras best\_model.keras for emotion classification, returning time-series emotion distributions and confidence scores (Phase 3A).
  3. **transcription.py** wraps OpenAI Whisper for robust speech-to-text conversion, handling audio extraction with FFmpeg and punctuation normalization (Phase 3B).
  4. **grammar\_checker.py** combines local LanguageTool checks with optional GPT-4 enhancements to detect spoken-language-specific grammar issues and propose improvements (Phase 3C).
  5. **candidate\_evaluator.py** implements rubric-based scoring for Technical and HR questions, leveraging FAISS retrieval, Azure OpenAI embeddings, and historical datasets , and applies weighted aggregation logic (Phase 4).
  6. **pdf\_report\_generator.py** produces polished, corporate-style PDF reports—embedding Matplotlib charts, confusion matrices, and executive summaries—via ReportLab (Phase 6).

Each component manages its own dependencies (model files in models/, external APIs) and returns structured dictionaries. This design aligns with the single‐responsibility principle, enabling independent testing and future replacement or extension of analysis modules.

* **Utility Modules (src/utils/)**  
  Common tasks are factored into utils/ to avoid duplication:
  1. **file\_utils.py** ensures directory existence and generates timestamped file paths for recordings and results.
  2. **config\_loader.py** reads config/settings.py to centralize file paths, API credentials, and interview question banks, exposing a singleton Config object.
  3. **logger.py** standardizes Python logging across components, configuring log levels and handlers for both console and file outputs.

By abstracting path management, configuration loading, and logging into reusable helpers, the system maintains consistent behavior and reduces boilerplate in both the Streamlit app and feature components.

**Figure 4.3.1** Project directory layout illustrating separation of UI (app.py), analysis components (components/), and shared utilities (utils/):

A screen shot of a computer program

AI-generated content may be incorrect.

This three-tiered organization underpins the system’s **scalability**—new analysis modules or UI enhancements can be added without impacting unrelated layers—and its **robustness**, as each module validates its inputs and handles failures gracefully before propagating results upstream.

**4.3.2 Emotion Analyzer**

The EmotionAnalyzer module implements a pipeline for automatically detecting and quantifying emotional states from a recorded video’s audio track. It relies on a pre-trained deep neural network for classification, combined with classical signal-processing techniques for robust feature extraction and segmentation. The key stages are:

**1. Model, Scaler, and Encoder Initialization**

Upon instantiation, the analyzer loads three essential artifacts:

1. **Keras classification model**
   * A TensorFlow/Keras architecture (e.g. a CNN or LSTM) trained on labeled speech data to predict discrete emotion categories.
   * Loaded via tensorflow.keras.models.load\_model(model\_path).
2. **StandardScaler**
   * A sklearn.preprocessing.StandardScaler fitted on training-set features to normalize input statistics (zero mean, unit variance).
   * Deserialized from scaler\_path with Python’s pickle.
3. **Label encoder**
   * A sklearn.preprocessing.LabelEncoder (or similar) that maps integer prediction indices back to human-readable emotion labels (e.g. “happy,” “sad”).
   * Also loaded via pickle from encoder\_path.

The analyzer fixes the following hyperparameters for feature dimensions:

* **Number of MFCC coefficients**: 40
* **Maximum frames per segment**: 100

**2. Audio Extraction from Video**

To isolate the speech signal, the analyzer invokes **FFmpeg** via a subprocess:

* Samples at 16 kHz, mono, 16-bit PCM.
* Verifies successful extraction by checking both the FFmpeg return code and the existence of the output file.

**3. Segmenting the Audio Signal**

Rather than analyzing the entire recording at once, the module performs coarse segmentation to isolate salient speech regions:

1. **Fixed-length slicing**: Splits the full waveform y into contiguous chunks of 6 s (chunk\_duration) each.
2. **Energy scoring**: Computes mean squared energy of each chunk and retains only the top 60 % by energy—thereby focusing on the most speech-rich segments.
3. **Temporal ordering**: Sorts the retained segments by their original start time to preserve chronology.

Each segment is represented as a dictionary containing:

* start, end times (in seconds)
* energy metric
* Raw audio samples (audio)

**4. Feature Extraction and Augmentation**

For each retained segment, the analyzer constructs a **stack** of five feature vectors to improve robustness via data augmentation:

1. **Base features** from the original waveform
2. **Additive noise**: Signal plus random uniform noise scaled to 3.5 % of the signal’s maximum amplitude
3. **Time stretching**: Lengthened or compressed by a factor of 0.8 via Librosa’s time\_stretch
4. **Time shifting**: Circularly shifted by up to ±5 000 samples
5. **Pitch shifting**: Shifted by +0.7 semitones via Librosa’s pitch\_shift

Each feature vector comprises:

* **MFCCs**: 40 coefficients, computed with librosa.feature.mfcc, then zero-mean/unit-variance normalized and either padded or truncated to exactly 100 frames.
* **Zero-crossing rate**: 1-dim feature array, similarly padded/truncated.
* **Root-mean-square energy**: 1-dim feature array, padded/truncated.

These three sources are **vertically concatenated** and flattened to a one-dimensional vector of length (40+1+1) × 100 = 4 200 per augmentation. The result is a NumPy array of shape (5, 4 200).

**5. Emotion Classification per Segment**

For each of the five augmented versions of a segment:

1. **Scaling**: Apply the pre-loaded StandardScaler to each feature vector, then reshape to (4 200, 1) for model compatibility.
2. **Prediction**: Call model.predict(...) to obtain a probability distribution over emotion classes.
3. **Argmax voting**: Convert each probability vector to a discrete class index; then take a majority vote across the five augmentations to produce a single predicted label per segment.
4. **Confidence**: Compute the mean of the maximum predicted probabilities (across augmentations) as the segment’s confidence score.

This process yields two lists—one of predicted emotion labels and one of confidence values—one entry per retained segment.

**6. Aggregate Emotion Scoring**

After classifying all segments, the analyzer synthesizes a global emotional profile:

* **Dominant emotion**: The mode of the per-segment predictions (most frequent label).
* **Average confidence**: Arithmetic mean of segment confidence scores, scaled to [0, 1].
* **Distribution**: A Counter-based histogram mapping each emotion label to its segment count.
* **Total segments**: The number of segments included in the final distribution.
* **Detailed lists** (all\_emotions, all\_confidences) for downstream inspection.

If no valid segments survive (e.g. silent recording), a default “unknown” result with zero‐confidence and empty distributions is returned.

**7. Resource Cleanup**

Finally, the temporary WAV file is removed to conserve disk space.

**Summary of Outputs**

The method analyze(video\_path) returns a dictionary:

A screen shot of a computer code

AI-generated content may be incorrect.

These metrics feed directly into the interview scoring framework (Section 4.3.2), where emotional confidence and positivity bias contribute 20 % of the candidate’s overall performance score.

**4.3.3 Candidate Evaluator Workflow**

The **CandidateEvaluator** module implements a sophisticated, retrieval‐augmented evaluation pipeline for scoring and explaining candidate answers to both Technical and HR interview questions. It combines historical performance data, domain‐specific rubrics, and large‐language‐model reasoning in a multi‐stage process, as detailed below.

**1. Initialization and Resource Loading**

Upon instantiation, CandidateEvaluator performs the following setup steps:

1. **Environment Configuration**
   * Loads Azure OpenAI credentials from .env via load\_dotenv().
   * Sets AZURE\_OPENAI\_API\_KEY, AZURE\_OPENAI\_ENDPOINT, and OPENAI\_API\_TYPE="Azure" environment variables.
2. **LLM Client Initialization**
   * Creates an AzureChatOpenAI client targeting the GPT-4O Azure deployment (“GPT-4O-50-1”) using API version 2023-12-01-preview.
3. **Rubric and Historical Data Ingestion**
   * Reads *Technical* and *HR* rubrics from JSON files (tech\_rubric.json, hr\_rubric.json).
   * Reads past evaluation results (tech\_evaluation\_results.json, hr\_evaluation\_results\_with\_samples.json) and constructs mappings

past\_scores[ question ]↦previous overall\_score \text{past\\_scores}[\,\text{question}\,]\mapsto \text{previous overall\\_score}past\_scores[question]↦previous overall\_score

for each domain.

1. **Corpus Indexing via FAISS**
   * Loads question–answer pairs from CSVs (dataset.csv, interview\_best\_answers\_samples.csv).
   * Builds vector‐embeddings for question texts using AzureOpenAIEmbeddings.
   * Constructs two FAISS‐based retrieval indexes (Technical & HR) with a k-nearest retrieval strategy (k=3).
2. **RetrievalQA and LLMChain Setup**
   * **Exact‐Match Chains**
     + For each domain, a RetrievalQA chain uses a custom prompt to ask:

“Does this *new* question exactly match one of these *retrieved* questions? Respond YES: “<matched question>” or NO.”

* + **Relevance Chains**
    - Two LLMChain instances (Technical & HR) prompt GPT-4O to filter the top-3 FAISS neighbors, returning a JSON array of those deemed truly relevant.

**2. Single-Answer Rubric-Based Evaluation (evaluate\_with\_rubric)**

To assess the quality of an individual response against a JSON‐defined rubric:

1. **Prompt Construction**
   * Encodes the rubric (an array of {name, description} entries), the interview question, and the candidate’s answer into a single LLM prompt instructing GPT-4O to output:

{

"scores": [

{ "name": <criterion>, "score": <0–100>, "explanation": <one-sentence> },

…

],

"overall\_score": <average of all scores>

}

1. **Multiple Independent Evaluations**
   * Runs the above chain **three** times to obtain three independent score+explanation JSON objects, mitigating single-pass noise.
2. **Score Aggregation**
   * For each rubric criterion:
     + Averages the three numeric scores (rounded to two decimals).
     + Summarizes the three one‐sentence rationales into a single concise rationale via a secondary GPT-4O prompt.
3. **Overall Score Computation**
   * Averages the per‐criterion averages to yield a final overall\_score.

This method yields both a fine‐grained **rubric\_breakdown** and a scalar **overall\_score** for the new answer.

**3. Answer Classification and Score Fusion (evaluate\_question\_answer)**

When presented with a (question, answer) pair, the evaluator proceeds as follows:

1. **Type Determination**
   * Maps the question text to its domain (Technical vs. HR) via a pre‐defined lookup.
2. **Trivial Answer Handling**
   * Answers of the form “I don’t know” or fewer than three words receive 0 across all rubric criteria immediately.
3. **Nearest‐Neighbor Retrieval**
   * Queries the FAISS index for top‐3 semantically similar questions.
   * Concatenates their texts into a context string for “exact‐match” checking.

**5. Key Functional Highlights**

* **Retrieval‐Augmented Exact Matching** ensures that known exemplar answers (with historical ground‐truth scores) inform evaluation when appropriate.
* **LLM‐Mediated Relevance Filtering** prunes semantically similar but contextually irrelevant neighbors, safeguarding against spurious score inheritance.
* **Multi‐Pass Rubric Consensus** (three independent LLM runs + rationale summarization) reduces stochastic variance in GPT‐based scoring.
* **Hybrid Score Fusion** dynamically blends historical consistency with fresh rubric‐based judgments, weighting past performance more heavily only when precedent quality is high.

Collectively, these capabilities deliver a robust, transparent, and data‐driven answer evaluation subsystem, tightly integrated with the broader interview analysis platform.

  4.3.4 Audio/Video Recording Workflow

The AudioVideoRecorder component encapsulates the complete lifecycle of multimedia capture—preview, synchronized audio/video recording, per‐question file management, and post‐processing—ensuring that each question’s response is individually tracked and stored (Fig. 4.3.2). This workflow unfolds in four principal stages: **camera preview initialization**, **recording orchestration**, **audio/video synchronization and cleanup**, and **question‐specific management**.

**1. Camera Preview Initialization**

Prior to any recording, the system allows the interviewer or candidate to verify camera connectivity and framing via a real‐time preview. Invoking start\_preview() attempts to open the default webcam (index 0) through OpenCV’s VideoCapture. If successful, the capture object is configured to a standard resolution (640×480 px) and frame rate (20 FPS) for consistent downstream processing. During preview, get\_frame() can be called at any time to retrieve the latest BGR frame, which is converted to RGB and displayed in the Streamlit UI. This preview loop continues until stop\_preview() is invoked, at which point the camera resource is released—unless a recording is in progress. Error handling at each step surfaces user‐friendly messages in case of permission denials or hardware failures.

**2. Recording Orchestration**

When the candidate is ready to respond, the UI triggers start\_recording(duration, question\_id), which performs the following:

1. **Question Identification & Path Generation**
   * Associates the upcoming recording with a numeric question\_id, enabling per‐question file management.
   * Constructs timestamped filenames for temporary video (temp\_video\_q{ID}\_YYYYMMDD\_HHMMSS.mp4), temporary audio (temp\_audio\_q{ID}\_YYYYMMDD\_HHMMSS.wav), and the final combined output (interview\_q{ID}\_YYYYMMDD\_HHMMSS.mp4) under data/recordings/.
2. **Threaded Capture Launch**
   * **Video Thread** (\_record\_video): Writes successive frames from the live VideoCapture into an OpenCV VideoWriter at 20 FPS. A loop guard tracks elapsed time to enforce the user‐specified maximum duration (e.g. 120 s).
   * **Audio Thread** (\_record\_audio): Opens a sounddevice.InputStream at 16 kHz mono, registering a callback to append each buffer of raw PCM samples into self.audio\_frames. This thread likewise respects the duration limit.

This dual‐threaded design ensures that audio and video capture proceed in parallel without blocking the main UI thread. Shared flags (self.recording) coordinate graceful shutdown upon manual stop or time expiration.

**3. Audio/Video Synchronization and Cleanup**

Upon completion—either via manual stop\_recording() or natural timeout—both threads are joined with short timeouts to ensure they terminate cleanly. The subsequent \_combine\_av() routine leverages MoviePy to:

1. **Load Temporary Clips**: Read the raw video and audio files via VideoFileClip and AudioFileClip.
2. **Duration Alignment**: Truncate both streams to the shorter duration, preventing drift or silent segments.
3. **Muxing**: Overlay the trimmed audio onto the video track and write the merged result as an H.264/AAC MP4, ensuring broad compatibility.
4. **Resource Management**: Close all clip objects and delete the temporary files to conserve disk space.

Error catches at each stage report any failures—such as missing files or codec mismatches—back to the UI log.

**4. Question‐Specific Management and Metadata**

Critically, the recorder maintains a question\_recordings dictionary that maps each question\_id to a metadata record:

* **file\_path**: Location of the final combined MP4.
* **timestamp**: Datetime of recording completion.
* **duration**: Measured via MoviePy post‐merge.

Utility methods—get\_question\_recording(), has\_question\_recording(), and get\_recording\_status()—allow the Streamlit app to query whether a given question has been captured, to retrieve its file for playback or analysis, and to present duration and file‐size metrics in the UI. Users may also delete individual recordings (delete\_question\_recording()) or clear all (clear\_all\_recordings()), facilitating re-takes without manual file‐system intervention.

**5. In-Frame Recording Feedback**

To provide candidates with clear visual cues during their response, the UI integrates a live timer overlay into each preview frame when self.recording is true. OpenCV primitives draw a semi-transparent rectangle and red text indicating elapsed or remaining seconds. A manual “Stop Recording” button breaks out of the capture loops immediately, updating both the on-screen indicator and the underlying recording flag.

  4.3.5 Transcription with Whisper

The Transcription component leverages OpenAI’s Whisper model to convert candidate responses from audio (or embedded video) into a textual transcript, forming the foundation for subsequent grammar and content analyses. Its design emphasizes robustness to variable audio quality, deterministic output, and minimal post‐processing to preserve natural speech patterns, including filler words and hesitations.

**1. Model Initialization**

Upon instantiation, Transcription(model\_name="base") invokes whisper.load\_model(model\_name), loading the selected pre‐trained Whisper variant into memory. Supported model sizes range from "tiny" through "large", trading off transcription speed against accuracy. Any exceptions during loading (e.g., missing weights or GPU memory exhaustion) are caught and re‐thrown after logging, ensuring that the calling application can gracefully disable downstream transcription if initialization fails.

**2. Audio‐Only Transcription (transcribe\_audio)**

This method accepts a path to a mono WAV file sampled at 16 kHz. It first verifies file existence, raising FileNotFoundError if missing. The core call to self.model.transcribe() is parameterized for deterministic, speech‐only transcription:

* **language="en"**  
  Forces English decoding, eliminating Whisper’s language‐detection overhead.
* **task="transcribe"**  
  Disallows translation, retaining original utterances.
* **temperature=0.0**  
  Uses greedy decoding for reproducibility.
* **no\_speech\_threshold=0.6**  
  Adjusts the model’s silence detector to avoid spurious “no speech” segments.
* **logprob\_threshold=-1.0** and **compression\_ratio\_threshold=2.4**  
  Control fallback behavior for noisy or repetitive audio.

The returned JSON contains a "text" field, which is stripped of leading/trailing whitespace and returned verbatim—preserving disfluencies for context‐aware grammar checking.

**3. Video‐Based Transcription (transcribe\_video)**

To handle candidate responses recorded as video files, transcribe\_video(video\_path) orchestrates an external audio‐extraction step followed by the above audio transcription:

1. **Temporary WAV Creation**  
   A tempfile.NamedTemporaryFile(suffix=".wav", delete=False) provides a unique filename for the extracted audio.
2. **FFmpeg Invocation**  
   A subprocess call to ffmpeg strips the video track (-vn), encodes PCM 16 kHz mono (-acodec pcm\_s16le -ar 16000 -ac 1), and applies a 2× volume boost (-af volume=2.0) to mitigate low‐level recordings. Errors in this step (e.g., missing ffmpeg binary or unsupported codec) raise a RuntimeError with the stderr payload.
3. **Delegated Audio Transcription**  
   The resulting WAV file is fed to transcribe\_audio().
4. **Cleanup**  
   The temporary file is unlinked immediately after transcription, ensuring no residual artifacts.

**4. Minimal Post‐Processing (\_clean\_transcript)**

Although Whisper outputs high‐quality text, the component includes a private \_clean\_transcript() stub designed for minimal sanitization:

* **Whitespace normalization**: Collapses multiple spaces.
* **Audio‐quality heuristic**: Flags transcripts with extreme repetition (over 80% identical tokens) as unreliable.

In practice, this cleaning step is deferred to the grammar checker to preserve candidate idiosyncrasies (e.g., “um,” “uh”) that are diagnostically informative.

**5. Integration and Error Handling**

All transcription operations are instrumented with print() or logger calls for debugging. Any exceptions during FFmpeg execution or model inference return a descriptive error string (e.g., "Transcription failed: <message>"), which the Streamlit UI surfaces to the user. The upstream perform\_analysis() function interprets such failures by:

* Displaying a warning that transcription was unavailable or unreliable.
* Skipping grammar and AI‐based evaluation steps if no transcript text is delivered.

**6. Role in the System Pipeline**

Within the overall workflow (Section 4.3.2), the Transcription component executes immediately after emotion analysis:

1. **Audio Extraction**: From the question‐specific video file.
2. **Text Generation**: Whisper produces the raw transcript.
3. **Grammar & Content Readiness**: The transcript is stored in analysis\_results['transcript'] for subsequent invocation of HybridGrammarChecker and CandidateEvaluator.

This modular separation ensures that audio issues (e.g., silence, noise, or codec errors) can be isolated and diagnosed without contaminating the performance of other components.

Through its careful configuration of Whisper parameters, tight error handling, and minimal post‐processing, the Transcription component provides reliable, high‐fidelity text output that underpins all downstream linguistic and semantic analyses in the AI‐powered interview system.

  4.3.6 Grammar Checking (LanguageTool + GPT)

The HybridGrammarChecker component employs a two-tiered “hybrid” approach to spoken-language grammar analysis: a local, rule-based pass via LanguageTool, followed by an optional, context-aware AI refinement via Azure OpenAI’s GPT. This design balances the reliability and immediacy of deterministic grammar rules with the deeper linguistic insights of a large language model, while intentionally **ignoring** orthographic and vocabulary critiques that are irrelevant to natural speech.

**1. Initialization and Capability Discovery**

Upon instantiation, the checker:

1. **Initializes LanguageTool** (language\_tool\_python.LanguageTool('en-US')), flagging any startup failure and setting local\_available=False if the library cannot be loaded.
2. **Probes Azure OpenAI** (if Config.is\_azure\_openai\_available()), configuring environment variables and instantiating AzureChatOpenAI. A lightweight “Test” message is sent to verify connectivity; on success, ai\_available=True.
3. **Establishes speech-aware filters** for punctuation, contractions, filler words, repetition, and sentence fragments, stored in self.speech\_filters.

The public method get\_analysis\_summary() reports these capabilities and recommends thresholds for AI triggering.

**2. Top-Level API: check\_grammar(text, force\_ai=False)**

This method orchestrates the full grammar-checking pipeline:

1. **Input validation**: Immediately returns an “empty” or “minimal” result if the text is blank or shorter than five words.
2. **Pre-processing** (\_clean\_text)
   * Collapses whitespace.
   * **Counts and removes** common speech fillers (e.g., “um,” “like,” “you know”), returning a cleaned transcript and a filler\_count.
   * Normalizes punctuation spacing around .?!.
3. **Local rule-based analysis** (\_check\_with\_language\_tool)
   * Executes LanguageTool’s check() on the cleaned text.
   * **Filters out** all spelling errors (by category, rule ID, and message heuristics).
   * **Ignores** overly formal or transcription-artifact violations (e.g., Oxford­-comma mandates, passive-voice suggestions, one-character tokens).
   * Records each remaining match with its category, severity (remapped for spoken registers), context, and up to three replacement suggestions.
   * Applies a **speech-aware correction** pass (\_speech\_aware\_correction) that only fixes high-confidence grammar rules such as subject-verb agreement.
4. **AI trigger decision** (\_should\_use\_ai)
   * Consults configuration flags (GRAMMAR\_AI\_AUTO\_TRIGGER, GRAMMAR\_AI\_THRESHOLD, AI\_TRIGGER\_ERROR\_RATE).
   * Triggers AI if the cleaned response exceeds a word‐count threshold (e.g., 30 words), or if the local error rate is high (e.g., > 8%), or if the response is very long (> 50 words).
5. **Optional GPT-powered refinement** (\_check\_with\_azure\_openai)
   * Crafts a tightly controlled prompt instructing the model to **ignore spelling and word choice**, focus exclusively on grammar (tenses, agreement, structure, clarity punctuation), and respond with a single balanced JSON object containing:
   * Extracts the JSON payload via a custom \_extract\_json\_from\_text routine; falls back to a canned “parsing error” result if extraction fails.
6. **Result composition**
   * If AI analysis succeeds, merges local and AI insights (\_merge\_results), computing a final grammar score as a weighted blend (40% local, 60% AI) and applying a filler-word penalty (up to 20 points).
   * If AI is unavailable or not triggered, finalizes purely local results (\_finalize\_local\_results), scoring via

score=100−(min⁡(errorswords×150,40)+min⁡(fillersoriginal\_words×100,20)) \text{score} = 100 - (\min(\tfrac{\text{errors}}{\text{words}}\times150,40) + \min(\tfrac{\text{fillers}}{\text{original\\_words}}\times100,20))score=100−(min(wordserrors​×150,40)+min(original\_wordsfillers​×100,20))

with a floor of 10 points.

1. **Post-processing outputs**
   * Returns a comprehensive dictionary containing:
     + **Core metrics**: grammar\_score, error\_count, filler\_count, filler\_rate, word\_count, sentence\_count.
     + **Local detail**: filtered local\_errors, corrected\_text.
     + **AI insights** (if used): key\_strengths, key\_issues, specific\_suggestions, interview\_assessment.
     + **Context flags**: analysis\_type ("local\_only", "hybrid", "empty", or "minimal"), ai\_used.
   * In the minimal or empty cases, stubbed scores and assessments convey that detailed feedback is unavailable.

**3. Role in the Interview Pipeline**

Within the broader Streamlit application (Section 4.3.2), HybridGrammarChecker is instantiated once per candidate response. Its outputs feed directly into:

* **Score aggregation** (calculate\_aggregate\_score), where grammar contributes 25 % weight to the overall interview performance.
* **UI presentation** (display\_grammar\_results), which highlights a composite Grammar Score, an overall assessment banner, AI-derived strengths/issues (in expanders), and a curated list of speech-relevant local errors.
* **PDF reporting**, where detailed grammar feedback appears alongside emotion and content analyses.

By combining deterministic rule-based filtering, speech-aware heuristics, and optional GPT-powered nuance, the HybridGrammarChecker delivers precise, context-appropriate grammar feedback tailored to spoken interviews—ensuring candidates receive actionable insights without undue focus on spelling, formality, or out-of-scope vocabulary.

  4.3.7 PDF Report Generation

**4.4 User‐Facing Functionalities**

The AI Interview System provides a rich, end-to-end interactive experience, guiding the user through practice questions, recording, live feedback, and comprehensive reporting. The main user‐facing capabilities include:

**4.4.1 Question Navigation & Session Management**

* **Randomized Question Selection**
  + At session start, the system randomly selects two Technical and one HR question (from a bank of eight) and shuffles their order.
* **Sequential & Direct Access**
  + **“Next”/“Previous” Buttons** allow linear progression through questions.
  + **Sidebar List** displays all questions with status icons (▶️ current, ✅ completed, ⏳ pending), and expanders to preview any upcoming question.
* **New Interview Reset**
  + A “🔄 New Interview” button clears all recordings and analysis state, letting the user restart with a fresh random set.

**4.4.2 Recording Controls & In‐Frame Timer**

* **Camera Preview**
  + Start/Stop camera controls show a live video feed.
* **Countdown & Automatic Limit**
  + A three-second visual countdown precedes each recording.
  + Recordings auto-stop at 2 minutes (configurable), with visual progress bars and an in-frame timer overlay.
* **Manual Stop**
  + The “⏹️ Stop Recording” button lets the candidate end recording early if they finish their answer before the time limit.
* **Recording Status Indicators**
  + Dynamic banners and progress bars clearly show when recording is active, stopped manually, or complete.

**4.4.3 Question‐Specific Analysis & Results**

Once a recording is analyzed, the user can:

* **Show Results**
  + A “📊 Show Results” button appears on the question card.
* **Detailed Breakdown**
  + **Emotion Analysis** 🎭: dominant emotion, confidence score, segment‐by‐segment distribution and progress bars.
  + **Transcription** 📝: full interview transcript in a scrollable text area.
  + **Grammar & Communication** 📝:
    - A speech‐aware grammar score (0–100), with severity‐filtered issues, AI-strengths and improvement suggestions in expanders.
  + **AI Answer Evaluation** 🤖: rubric‐based scores with criterion‐level breakdowns and one-sentence explanations, organized into styled cards or expanders.
* **Re-record Option**
  + A “🔄 Re-record” button clears that question’s data, allowing a fresh take.

**4.4.4 Aggregate Session Summary**

After completing one or more questions, the user can view a **Complete Interview Results Summary**:

* **Progress Metrics**
  + Total questions vs. completed count, plus a completion percentage.
* **Overall Performance**
  + A single “Aggregate Score” out of 100, color-coded (green/yellow/red), with descriptive badges (“Excellent”, “Good”, etc.).
* **Component Breakdown**
  + Side-by-side cards for Emotional Intelligence, Communication Skills, and Content Quality, each showing score, weight, and sub-descriptions.
* **Visual Charts**
  + Progress bars and metrics for each component for at-a-glance insight.
* **Per-Question Expanders**
  + Collapsible sections for every question with its full detailed results, as above.

**4.4.5 Practice Mode & Question Regeneration**

* **“New Interview” Practice**
  + At any time, the user may click “🔄 New Interview” to generate a fresh set of practice questions.
* **Balanced Coverage**
  + The system ensures a mix of Technical and HR prompts each session, supporting varied practice.

**4.4.6 On-Demand PDF Reporting**

* **Per-Question PDF**
  + A “📄 PDF” button next to each question’s results generates a focused report (analysis, transcripts, metrics).
* **Complete Interview Report**
  + A “📄 Generate Complete PDF Report” button compiles all results, scores, charts, and verbatim transcripts into a single multi-page PDF.
* **Download Buttons**
  + Embedded Streamlit download widgets deliver the PDFs directly to the user’s local machine.

**4.4.7 Key UX Highlights**

* **Responsive Feedback**: Real-time banners, progress bars, and camera overlays keep the candidate informed at every step.
* **Flexible Control**: Manual recording stop and re-recording give users control over pacing.
* **Actionable Insights**: Emotion metrics, grammar suggestions, and rubric explanations turn raw recordings into concrete learning opportunities.
* **Portability**: Exportable PDFs allow offline review, coach sharing, or portfolio inclusion.

Together, these features form a cohesive, user-centered practice environment—transforming raw mock interviews into guided learning experiences with data-driven feedback at every stage.

 4.5 Technologies Used  
 **4.5.1 Streamlit, LangChain & FAISS**

* **Streamlit**
  + *Role:* Serves as the primary web-app framework, driving our interactive interview UI, session state management, and real-time orchestration of capture, analysis, and reporting phases.
  + *Key Features:*
    - Declarative components (st.button, st.columns, st.session\_state) for rapid prototyping of responsive layouts and widgets.
    - Built-in caching (@st.cache\_data, @st.cache\_resource) to avoid redundant model loads and accelerate repeated inferences.
    - Download widgets (st.download\_button) to stream PDF/HTML reports directly to users.
  + *Integration:* The app.py entrypoint imports each analysis component (e.g., emotion\_analyzer, candidate\_evaluator) and wires them into a five-phase workflow—Setup ▶ Recording ▶ Analysis ▶ Aggregation ▶ Reporting—using Streamlit callbacks and session variables.
* **LangChain**
  + *Role:* Orchestrates multi-step LLM interactions in our answer-evaluation pipeline, including dynamic rubric generation, exact-match checks, relevance filtering, and rationale summarization.
  + *Key Features:*
    - Chain abstractions (LLMChain, RetrievalQA) that simplify prompt-template management and batch execution.
    - Support for multiple prompt stages, output parsers, and memory buffers.
  + *Integration:* We define a RubricGenerationChain to batch-produce and post-process per-question rubrics, and two RetrievalQA chains (Technical & HR) to perform “exact match?” and “relevance check?” tasks over FAISS neighbors.
* **FAISS**
  + *Role:* Provides high-performance, in-memory vector indexing and nearest-neighbor search for our HR and Technical question banks.
  + *Key Features:*
    - IndexFlatL2 for exact L2 similarity searches over dense embeddings.
    - Python bindings for sub-millisecond retrieval over thousands of vectors.
  + *Integration:* We embed all historical questions with AzureOpenAIEmbeddings and load them into separate FAISS indices. At evaluation time, each new question’s embedding is queried (k=3) to retrieve semantic neighbors for score-fusion logic.

**4.5.2 GPT-4o & Retrieval-Augmented Generation (RAG)**

* **GPT-4o (AzureChatOpenAI)**
  + *Role:* Acts as the core generative engine for:
    1. Rubric criterion creation (5–8 items per question),
    2. Triplicate rubric-based scoring (0–100) with rationale,
    3. Rationale consolidation, and
    4. Final “no-match” full evaluations.
  + *Key Features:*
    1. Advanced reasoning, few-shot learning via system↔user↔assistant prompt flows.
    2. Streaming output support for partial-result UIs.
  + *Integration:* Instantiated via AzureChatOpenAI(deployment\_name="GPT-4O-50-1") and passed to all rubric and evaluation chains in LangChain.
* **Retrieval-Augmented Generation (RAG)**
  + *Role:* Ensures that each evaluation is grounded in concrete examples by injecting top-K FAISS-retrieved Q/A pairs into the LLM prompt.
  + *Mechanism:*
    1. Retrieve top-3 neighbors,
    2. Filter for exact semantic match (70:30 or 100% rubric blend),
    3. Otherwise perform relevance filtering via a second LLM pass,
    4. Blend historical scores (30:70) when relevant neighbors exist.
  + *Benefits:* Reduces hallucination risk, stabilizes scoring variance, and leverages past validated evaluations.

**4.5.3 Whisper Speech-to-Text**

* *Role:* Transcribes candidate audio responses into text, forming the foundation for grammar checking and LLM-based content evaluation.
* *Model Details:*
  + Uses OpenAI’s whisper.load\_model("base"), encoder-decoder transformer pretrained on 680 K hours of multilingual, noisy data.
  + Parameters: language="en", task="transcribe", temperature=0.0 for deterministic outputs.
* *Integration:* The Transcription component invokes Whisper on extracted WAV streams (via FFmpeg), then applies minimal post-processing (\_clean\_transcript) before passing results to the HybridGrammarChecker and CandidateEvaluator.

**4.5.4 TensorFlow/Keras for Speech Emotion Recognition (SER)**

* *Role:* Trains and runs our 1D-CNN SER model, transforming raw audio waveforms into emotion class probabilities.
* *Key Features:*
  + Sequential API with five Conv1D blocks (512→512→256→256→128 filters), batch normalization, progressive dropout (0.3→0.5), and global pooling.
  + Custom callbacks: delayed EarlyStopping (start after epoch 40), ReduceLROnPlateau, and checkpointing on val\_accuracy.
* *Integration:*
  + Preprocessing in TensorFlow (tf.data.Dataset) with on-the-fly augmentations (noise, time-stretch, shift, pitch).
  + Model compiled with Adam (lr=1e-3) and categorical\_crossentropy.
  + Final prediction uses majority-vote over augmented segments to produce a dominant emotion and confidence score.

**4.5.5 Additional Tools**

* **OpenCV**
  + *Role:* Manages real-time camera preview and video capture (Phase 2), overlays timers, and synchronizes audio/video streams.
* **MoviePy**
  + *Role:* Combines separate audio and video threads into final MP4s, ensuring H.264/AAC compatibility and duration alignment.
* **LanguageTool**
  + *Role:* Performs rule-based grammar checks on transcripts, filtering filler words and non-speech artifacts before hybrid AI refinement.
* **Matplotlib & Seaborn**
  + *Role:* Generates learning curves, confusion-matrix heatmaps, and other diagnostic plots during development and in PDF reports.
* **ReportLab**
  + *Role:* Composes multi-page, corporate-style PDF reports embedding charts, tables, and executive summaries.

Together, these technologies form a cohesive, modular stack that delivers scalable, objective, and richly detailed feedback on both the emotional and content dimensions of video-interview responses.

 4.6 Trials & Experiments  
  4.6.1 SER Model Training Experiments

Ravdess+TESS

**Trial 1 – Transformer + LSTM (Baseline Augment-Noise)**

* **Architecture:**
  + 2-layer nn.TransformerEncoder (d\_model = 40, nhead = 4, FFN = 512)
  + 2-layer LSTM (hidden\_size = 128)
  + Final dense → softmax
* **Preprocessing & Augmentations:**
  + MFCC(40) padded/truncated to 400 frames
  + Random additive Gaussian noise (30 % chance)
  + Noise reduction via noisereduce
* **Result:**
  + **Test Accuracy:** 76.6 %
  + Macro F₁ ~ 75 %

**Trial 2 – Transformer + LSTM (Phase-Vocoder Stretch)**

* **Differs from Trial 6 →** uses librosa.phase\_vocoder for time-stretch augmentation (30 % chance) instead of random noise.
* **Result:**
  + **Test Accuracy:** 68.2 %
  + Drop in performance suggests additive noise + noise-reduction was more effective.

**Trial 3 – Transformer + LSTM (No Augment)**

* **Differs →** only MFCC + noise-reduction, no random augmentations.
* **Result:**
  + **Test Accuracy:** 74.6 %
  + Removing augmentation hurts generalization vs. Trial 6 (76.6 %).

**Trial 4 – Transformer + LSTM (Shift-Only Augment)**

* **Differs →** uses only random time-shifting (roll) augmentation (30 % chance), no noise addition.
* **Result:**
  + **Test Accuracy:** 72.1 %

**Trial 5 – Transformer + LSTM (Combined Augment: Noise + Shift)**

* **Differs →** both noise addition and random shift applied (30 % each), plus noise reduction.
* **Result:**
  + **Test Accuracy:** 77.8 % ⋙ **best of the Transformer + LSTM variants**

RAVDESS + TESS

**Trial 6: Conv1D + BiLSTM w/ ROS oversampling & focal loss**  
• **Datasets:** RAVDESS + CREMA-D + TESS + SAVEE  
• **Preprocessing:** MFCC (40×100) + light pitch/time augment (30% chance)  
• **Balancing:** RandomOverSampler → equal class counts  
• **Loss:** Focal loss (α per class from inverse train freq, γ=2)  
• **Split:** 80 / 20 train/val (6 784 / 1 696)  
• **Best train acc:** 85.8 % **Best val acc:** 83.8 %  
• **Notes:** Solid mid-range performance; benefits from balancing and focal loss.

**Trial 7: Baseline BiLSTM**  
• **Datasets:** RAVDESS + TESS  
• **Preprocessing:** MFCC (40×100), no label harmonization  
• **Architecture:** BiLSTM → BN → Dropout → Dense  
• **Balancing:** none  
• **Callbacks:** EarlyStopping, ReduceLROnPlateau, batch-detail logger  
• **Results:** 85.70 % train / 85.02 % val accuracy

**Trial 8: Harmonized Labels + BiLSTM**  
• **Diff vs. 16:** applied label mapping (fear→fearful, ps/pleasantsurprise→surprised)  
• **Results:** 85.70 % train / 84.02 % val accuracy

**Trial 9: Class-Weighted BiLSTM**  
• **Diff vs. 17:** computed balanced class\_weight and passed to fit  
• **Results:** 79.60 % train / 77.95 % val accuracy

**Trial 10: Upsample “calm” + Focal-Loss BiLSTM**  
• **Diff vs. 18:** up-sampled “calm” to majority count + focal loss (α=0.25, γ=2)  
• **Results:** 82.99 % train / 79.60 % val accuracy

**Trial 11: Conv1D → BiLSTM + Full Upsample + Focal Loss**  
• **Diff vs. 19:** added Conv1D front-end; up-sampled **all** classes; focal loss (α from orig dist)  
• **Results:** 80.20 % train / 76.36 % val accuracy

Ravdess-TESS-SAVEE,Crema-D

**Trial 12: Baseline (80/20 Split, Minimal Dropout)**

* **Setup**
  + 80% train / 20% test split
  + Conv1D network with five convolutional blocks (512→512→256→256→128 filters), minimal dropout (0.2)
  + No dedicated validation set
* **Results**
  + **Train Accuracy:** 0.9999
  + **Validation (Test) Accuracy:** 0.9252
* **Challenges**
  + **Overfitting**: Near-perfect training accuracy but a ~7 pp gap to test
  + **No Early Validation**: Risk of “validation leakage”

**Trial 13: Hold-Out Validation (90/5/5 Split, Higher Dropout)**

* **Setup**
  + 90% train, 5% validation, 5% test
  + Increased dropout rates in each block: 0.3, 0.4, 0.4, 0.5, 0.5 and 0.5 in the final dense layer
  + Delayed early stopping (only after epoch 40) to allow learning stabilization
* **Results**
  + **Train Accuracy:** 0.9996
  + **Validation Accuracy:** 0.8956
  + **Test Accuracy:** 0.8951
* **Challenges & Solutions**
  + **Under-fitting Tendency**: Higher dropout caused a drop in test performance
  + **Solution**: Tuned dropout per block to balance regularization vs. capacity

**Trial 14: True Validation vs. Hold-Out (80/8/12 Split)**

* **Setup**
  + 80% train → split further 10% for validation (yielding ~72/8/20 overall)
  + Moderate dropout (0.2 per block)
* **Results**
  + **Train Accuracy:** 0.9999
  + **Validation Accuracy:** 0.9240
  + **Test Accuracy:** 0.9240
* **Observations**
  + **Closer Alignment** between validation and test metrics, indicating a faithful early-stopping signal
  + **Remaining Overfitting**: Slight gap remained between train (≈100%) and test (≈92.4%)

**Trial 15: L₂-Regularization Instead of Dropout**

* **Setup**
  + Same data split as Trial 4 (90 % train, 5 % val, 5 % test)
  + All dropout layers removed
  + Added L₂ weight decay (kernel\_regularizer=l2(1e-4)) on every convolutional and dense layer
  + Callbacks:
    - ModelCheckpoint (save best on val\_accuracy)
    - ReduceLROnPlateau (halve LR on plateau)
    - Custom EpochDetailLogger (per-epoch metrics)
* **Results**
  + **Train Accuracy:** 0.9999
  + **Validation Accuracy:** 0.9099
  + **Test Accuracy:** 0.9132
  + **Per-Class Performance:** macro-F₁ ≈ 93 %; seen in the confusion matrix above.
* **Observations**
  + L₂ alone provided some regularization, but **under-performed** the carefully tuned dropout schedule of Trial 4 (Test 95.4 %).
  + Validation/test accuracy (~91.3 %) improved over the naive dropout of Trial 2 (~89.5 %), but still trailed the best configuration.

**Trial 16: Final Regularization Tuning (90/5/5 Split, Optimized Dropout)**

* **Setup**
  + Re-adopted the 90/5/5 split with final dropout schedule:
    - Block 1: 0.3
    - Block 2: 0.4
    - Block 4: 0.4
    - Block 5: 0.5
    - Dense: 0.5
  + DelayedEarlyStopping (patience=5 after epoch 40) and ReduceLROnPlateau
* **Results**
  + **Train Accuracy:** 0.9998
  + **Validation Accuracy:** 0.9493
  + **Test Accuracy:** 0.9543
* **Outcome**
  + **Generalization Improved** significantly, with test performance matching validation and minimal overfitting.
  + **Confusion Matrix & F-scores**: Balanced performance across all eight emotion classes (macro-F1 ≈96%).

  5.6.2 Rubric Generation Trials  
  5.6.3 Alternative Whisper Models  
  5.6.4 Embedding & Vector Database Tests  
  5.6.5 Prompt Engineering Approaches

6 Testing & Evaluation  
 6.1 Testing Methodology  
 6.2 Evaluation Metrics & Procedures

7 Results & Discussions  
 7.1 SER Accuracy & Confusion Matrix  
 7.2 Example Answer Evaluations  
 7.3 Sample Interview Sessions  
 7.4 RAG-Based & Human Evaluation Insights

8 Conclusions & Future Work  
 8.1 Summary of Achievements  
 8.2 Potential Improvements & Extensions

References

Appendix I Additional Code Snippets & Training Logs  
Appendix II Configuration Files & Paths  
Appendix III Extra Figures & Tables  
Appendix IV Work Plan Gantt Chart