Abstract

We present an end-to-end system for automated analysis of live interview recordings. The pipeline begins with a speaker diarization module that segments a multi-speaker audio recording into labeled speaker turns. A lightweight speaker mapping module then assigns these turns to roles (e.g., Interviewer vs. Candidate) based on a first-speaker-lock heuristic. Next, each audio segment is processed through an Automatic Speech Recognition (ASR) engine (OpenAI's Whisper) and a Speech Emotion Recognition (SER) model (Sense-Voice-Small) to produce text transcripts augmented with emotion labels. The final output is a structured JSON transcript that links speaker identity, time intervals, utterance text, and emotion. This automated approach aims to streamline the interview workflow by providing objective transcripts and emotional cues, thereby reducing reliance on manual notes and mitigating human bias. We detail the system architecture and evaluate its components using metrics such as diarization error rate (DER) and SER accuracy. Experimental results on representative interview and dialogue datasets demonstrate the effectiveness of our pipeline in accurately segmenting speakers and recognizing emotional tone, highlighting potential for fairer and more consistent interview assessment.

Chapter 2
Introduction

Automated analysis of interviews has gained attention as organizations seek objective and scalable evaluation methods. Traditional human-led interviews can be inconsistent and subject to bias, motivating the use of machine learning to assist assessment. In this paper, we propose an end-to-end system that processes interview recordings, extracts structured information, and generates candidate performance assessments. The system leverages recent advances in speech recognition, speaker diarization, and large language models (LLMs) to mimic aspects of human evaluators. The core contribution is a pipeline that segments conversation by speaker and topic, retrieves relevant evaluation criteria, and produces a detailed feedback report. By combining multiple AI components, the proposed system aims to provide consistent, data-driven interview analysis.

Chapter 3
Motivation

Interviews are a key element of hiring, yet they often suffer from subjectivity and inconsistency. Standardizing evaluation criteria is challenging when different interviewers have varying styles. An automated interview assessment system can address these issues by providing uniform analysis and reducing human biases. Moreover, with the growth of remote and large-scale hiring processes, manual assessment becomes impractical. Our motivation is to develop a system that not only transcribes spoken responses but also understands context and emotion. Such a system can aid recruiters by highlighting strengths and weaknesses of candidates in terms of communication skills, technical content, and behavioral fit. Additionally, embedding ethical design and fairness considerations ensures the system's recommendations do not reinforce bias. This work focuses on building an integrated architecture that handles audio processing, semantic analysis, and knowledge-based evaluation.

Chapter 4
Overview of Whisper Models

4.1 Whisper-Streaming: Real-Time Speech Recognition

4.1.1 Overview

Whisper-Streaming is a significant advancement in real-time speech recognition. [1] It transforms the high-performing OpenAI Whisper model into a streaming transcription engine, capable of converting speech into text with impressive speed and accuracy. [1] The system's innovation lies in its *Local-Agreement policy* combined with *self-adaptive latency*, striking a balance between transcription responsiveness and precision [1]

4.1.2 Core Components and Their Roles

Update Loop

At the core of Whisper-Streaming is its update loop, which continuously receives and processes incoming audio chunks. The loop adapts its processing frequency based on system capabilities to maintain optimal performance. [1] A key parameter, `MinChunkSize`, determines the minimum length of audio processed in each iteration, allowing a trade-off between latency and transcription accuracy. [1]

Skipping Confirmed Parts

To avoid redundant computation, Whisper-Streaming implements a confirmation-based skipping mechanism. [1] It compares unprocessed audio with previously confirmed words by analyzing timestamps within a one-second interval. Using n-gram matching ($n = 1$ to 5), confirmed words that match with new input are skipped, improving both speed and consistency. [1]

Audio Buffer Management

The system uses an audio buffer as temporary storage for incoming speech. [1] This buffer allows Whisper to process coherent segments rather than fragmented input. [1] It maintains a buffer length of up to 30 seconds, consistent with Whisper's original training context. [1] Once sentence-ending punctuation is detected in the confirmed output, the buffer is trimmed to maintain efficient operation. [1]

Inter-Sentence Context Joining

Whisper-Streaming ensures semantic continuity between sentences by incorporating inter-sentence context. [1] It extracts the last 200 confirmed words and passes them as a prompt in the next input to Whisper. [1] This strategy preserves context, style, and terminology across sentence boundaries, avoiding semantic drift and enhancing comprehension. [1]

4.1.3 Evaluation Dataset: ESIC Corpus

The Whisper-Streaming model is evaluated using the ESIC (European Parliament Speech Interpretation Corpus). The development set comprises 179 documents totaling approximately 5 hours of content, consisting of:

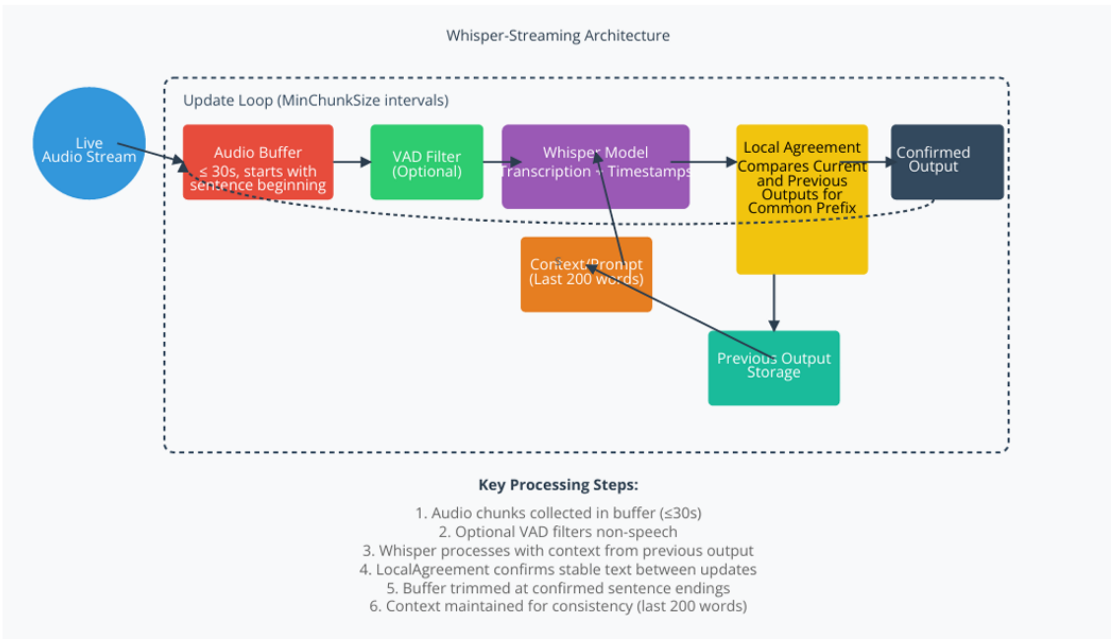
- Original English parliamentary speeches.
- Simultaneous interpretations in German and Czech.
- Manually annotated transcripts with word-level timestamps.

This corpus is used to benchmark both Word Error Rate (WER) and latency performance of real-time transcription systems. The evaluation was performed on an NVIDIA A40 GPU with 48GB memory.

4.1.4 The LocalAgreement-2 Policy

Stable Output Confirmation

The LocalAgreement-2 policy is a core innovation enabling real-time transcription. [1] It confirms transcribed text only when two consecutive outputs agree on the longest


Whisper-Streaming Architecture
Update Loop (MinChunkSize intervals)
Live Audio Stream
Audio Buffer ≤ 30s, starts with sentence beginning
VAD Filter (Optional)
Whisper Model Transcription + Timestamps
Local Agreement Compares Current and Previous Outputs for Common Prefix
Confirmed Output
Context/Prompt (Last 200 words)
Previous Output Storage

Key Processing Steps:

1. Audio chunks collected in buffer (≤30s)
2. Optional VAD filters non-speech
3. Whisper processes with context from previous output
4. LocalAgreement confirms stable text between updates
5. Buffer trimmed at confirmed sentence endings
6. Context maintained for consistency (last 200 words)

Figure 4.1: Architecture and workflow of Whisper-Streaming system

common prefix. [1]This strategy ensures that only stable and confident text segments are finalized, significantly reducing transcription flickering or retractions in live applications [1]

Streaming Optimization

The policy is tailored for continuous streaming scenarios, enabling incremental processing without the need to buffer entire sentences. [1] It achieves real-time performance by avoiding long waiting times typically required to reach sentence boundaries, making it suitable for applications such as interviews, live meetings, and simultaneous translation. [1]

Voice Activity Detection (VAD)

Whisper-Streaming provides flexibility with its VAD configuration:

- **VAD OFF**: Minimizes latency for uninterrupted, fluent speech. [1]
- **VAD ON**: Enhances accuracy during interpreted or naturally paused speech, though with an added latency of approximately 0.1–0.4 seconds. [1]

4.2 Simul-Whisper

4.2.1 Chunk-Based Streaming ASR Solution

Simul-Whisper is a novel adaptation of the pre-trained Whisper model tailored for streaming automatic speech recognition (ASR) without requiring fine-tuning. [2] It addresses key challenges in applying Whisper for real-time transcription tasks, making it suitable for applications like interviews and meetings [2]

4.2.2 Improvements Over Whisper-Streaming

Simul-Whisper directly addresses two major limitations of prior streaming efforts:

- **Encoder-Decoder Challenge**: Whisper's original encoder-decoder architecture is designed for full-input transcription, making it incompatible with real-time streaming scenarios where audio is incrementally processed. [2]
- **Chunk Truncation Issues**: Naively segmenting audio into chunks often causes errors at the chunk boundaries, such as repeated tokens or incomplete words. [2]

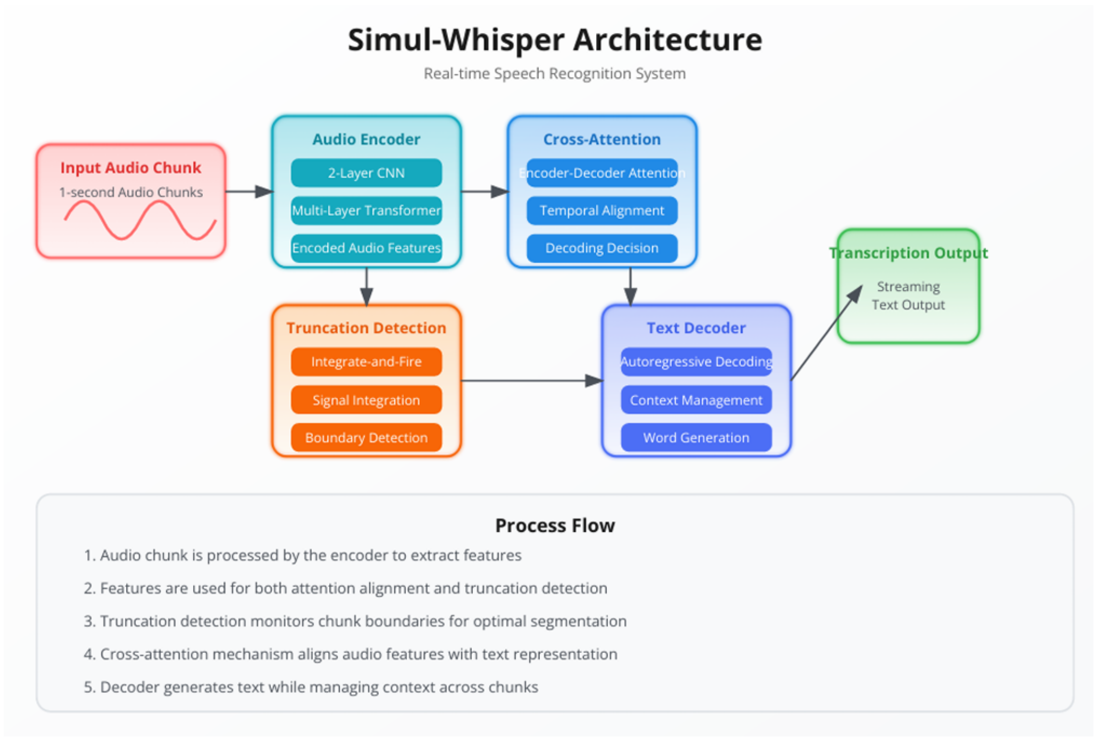
4.2.3 Core Methodology

Cross-Attention-Guided Decoding

Simul-Whisper uses Whisper's built-in cross-attention to effectively align acoustic features with decoder outputs. [2] This alignment guides when to stop decoding each chunk, minimizing the effects of chunk truncation. [2]

Integrate-and-Fire (IF) Truncation Detection

Inspired by biological neural firing models, an Integrate-and-Fire module accumulates encoder activations over time. [2] When a predefined threshold is reached, it marks the presence of a complete word. [2] If chunk truncation is detected during decoding, the partial or unstable transcription is discarded. [2]



Process Flow

1. Audio chunk is processed by the encoder to extract features
2. Features are used for both attention alignment and truncation detection
3. Truncation detection monitors chunk boundaries for optimal segmentation
4. Cross-attention mechanism aligns audio features with text representation
5. Decoder generates text while managing context across chunks

Figure 4.2: Architectural pipeline of Simul-Whisper

4.2.4 Data Evaluation

- **Datasets**: Simul-Whisper is evaluated on the LibriSpeech and Multilingual LibriSpeech (MLS) corpora.
- **Metrics**: Evaluation metrics include Word Error Rate (WER) and Differentiable Average Lagging (DAL) to quantify both transcription quality and latency.

4.2.5 Results

Superior Performance: Simul-Whisper achieves a minimal degradation of only 1.46% WER using 1-second chunks—significantly outperforming Whisper-Streaming's LocalAgreement policy under similar latency conditions.

4.2.6 Architectural Pipeline

The pipeline includes:

1. Chunked audio input.

2. Preprocessing and feature extraction.

3. Integrate-and-Fire truncation detector. [2]

4. Cross-attention decoding mechanism. [2]

5. Output filtering and token stabilization. [2]

4.2.7 Limitations and Future Work

- **Latency Reduction:** Current padding techniques introduce latency. [2] The authors explore self-distillation methods to minimize latency without compromising accuracy. [2]

- **Real-Time Processing:** Future versions aim to eliminate padding entirely through continuous audio streaming, improving responsiveness for live transcription scenarios. [2]

4.3 Whisper-T for Streaming Speech Processing (SSP)

Streaming Speech Processing (SSP) demands real-time transcription, posing challenges for speech foundation models like OpenAI's Whisper. [3] These challenges include fixed input lengths requiring padding, high encoding costs due to transformer layers, and inefficient decoding processes. [3] Whisper-T addresses these issues through optimizations like **hush words**, **beam pruning**, and **CPU/GPU pipelining**, aiming for sub-second latency and energy efficiency with minimal accuracy loss. [3]

4.3.1 Addressing SSP Challenges with Whisper-T

- **Fixed Input Length:** Models trained on 30-second audio clips require padding for shorter inputs, leading to redundant computation. [3]

- **High Encoding Cost:** Processing 1,500 tokens through transformer layers incurs high computational cost (GFLOPS), approximately 570 for 30-second padding. [3]

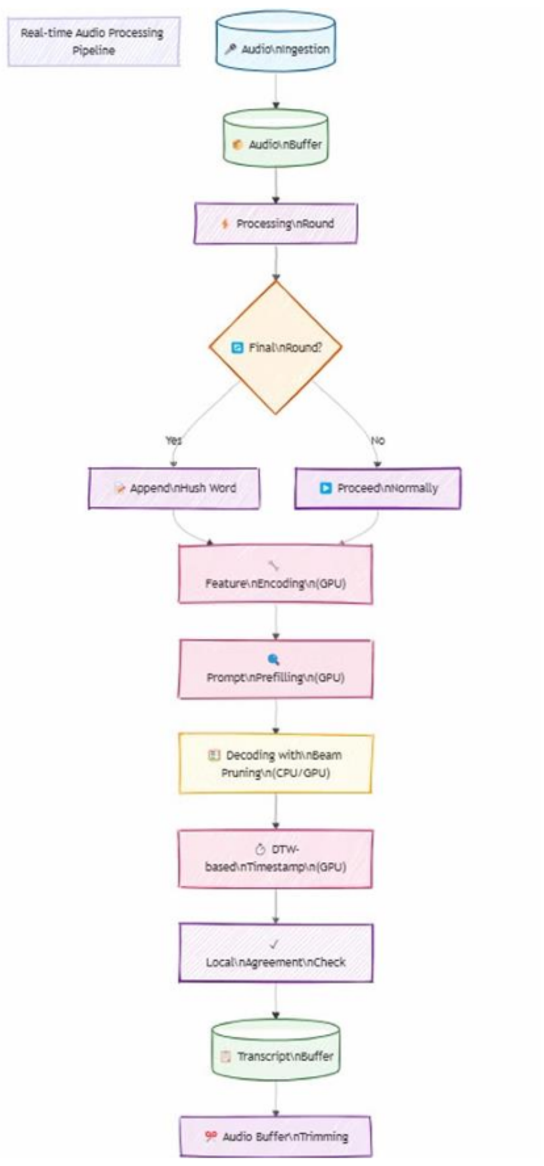


Figure 4.3: WhisperT Flowchart

- **Inefficient Decoding:** Beam search with 5 hypotheses is irregular, sequential, and GPU-unfriendly, increasing decoding latency. [3]

4.3.2 Key Components

Hush Words: Reducing Redundancy

Whisper-T replaces redundant padding with a short audio segment called a *hush word* to signal transcription termination. [3] This segment is appended to raw audio to avoid confusing the model, with an optimal length of 0.5 seconds. [3] Trained on datasets like LibriSpeech, this method:

- Reduces encoding GFLOPS by **3x**. [3]
- Minimizes hallucinations compared to naive padding methods. [3]

Beam Pruning: Reusing Intermediate Results

Beam pruning reduces redundant computation in beam search by reusing intermediate results. [3] Challenges include:

- Misaligned buffers. [3]
- Special tokens disrupting speculative decoding. [3]

Solutions involve search-based alignment and selective token matching. Results:

- Average beam size reduced from 5 to approximately 2.26. [3]
- Decoding time reduced by **30–50%** [3]

CPU/GPU Pipelining: Optimizing Resource Utilization

This technique optimizes resource utilization by mapping:

- **GPU tasks:** Encoding and prompt prefilling. [3]
- **CPU tasks:** Beam search decoding. [3]

With opportunistic offloading and dynamic thread allocation, the system:

- Reduces latency by **1.6x to 4.7x**. [3]
- Increases power consumption as a tradeoff. [3]

Chapter 5
Background

This section reviews relevant technologies: speaker diarization, automatic speech recognition (ASR), speech emotion recognition (SER), topic segmentation, and retrieval-augmented generation.

5.1 Speaker Diarization

Speaker diarization determines “who spoke when?” in an audio recording. Modern diarization systems often use neural pipelines with voice activity detection, speaker change detection, and clustering. We adopt a toolkit similar to pyannote.audio, which provides trainable neural blocks for this task. As shown in Figure 7.1, the raw interview audio is fed into a diarization model (e.g., Py-Annote) that outputs time-stamped speech segments for each speaker. A simple mapping heuristic then assigns speaker IDs to roles (e.g., interviewer vs. candidate) based on the first speaker rule or a short known prompt. Accurate diarization is crucial for attributing speech content to the correct participant in subsequent steps.

5.2 Speaker Mapping

Unlike speaker identification systems which recognize known individuals by embeddings verification, diarization only clusters speech segments by speaker consistency, it does not inherently know who each speaker is.

A key limitation of traditional diarization pipelines is that speaker labels (e.g., SPEAKER-00, SPEAKER-01, etc.) are assigned arbitrarily and non-deterministically. This means,

The same person (e.g., the interviewer) may be labeled differently across different audio files. Labeling may switch mid-session due to overlapping speech.
In evaluation scenarios (such as interviews), consistent role labeling (e.g., identifying the interviewer/s and the candidate) is crucial for downstream analysis where role has to be well defined.

The First-Speaker Lock technique ensures consistent speaker labeling in diarization by enforcing that the first speaker to appear in an audio file is always assigned the same label (e.g., ”Interviewer”). It enables ’Deterministic Labeling’ where the speaker who starts the conversation is always labeled ”Interviewer”. The other speaker becomes ”Guest”.
It is incredible in scenarios like,
1. Podcasts/Interviews: Where speaker roles are fixed (host/guest).
2. Multi-file Consistency: When processing chunks of the same conversation.

5.3 Automatic Speech Recognition and Emotion Recognition

After diarization and speaker-role mapping, each speech segment is processed by an ASR module to generate transcripts. We utilize a large-scale pre-trained ASR model (e.g., OpenAI’s Whisper) for robust transcription in diverse acoustic conditions. Whisper and similar models have demonstrated low word error rates (WER) across languages without extensive fine-tuning. Alongside ASR, we apply a speech emotion recognition (SER) component to tag each utterance with an emotion label (e.g., neutral, happy). We employ a foundation model such as SenseVoice, which jointly performs ASR and emotion analysis in a non-autoregressive manner. SenseVoice provides rapid inference and multi-lingual support, making it suitable for real-time interview scenarios.

5.4 Topic Segmentation

The interview is not just a bag of words, it's a structured narrative with evolving topics. The assessment of any given point should be sensitive to what was discussed before and how the conversation is flowing, that's where the idea of identifying shift in topics arises from.

Interviews often cover multiple topics (e.g., background, technical questions, teamwork). Identifying topic boundaries and understanding coherence, depth, transitions, recall, and completeness within and across topics is key to a nuanced assessment and helps contextualize candidate responses.

We perform topic segmentation on the cleaned transcript using an LLM-based classifier. The transcript is first pre-processed into a list of dialogue-turn objects (speaker, start/end times, text, emotion). Then, a prompted LLM outputs topic labels for contiguous segments of turns. This approach resembles few-shot classification, where the model assigns topic IDs based on content. The result is a topicSegment object list, each with a topic label and the indices of the dialogue turns in that segment. This step enables downstream modules to focus on specific topics independently, as depicted in Figure 8.2.

5.5 Retrieval-Augmented Generation (RAG)

To generate evaluations, we use a retrieval-augmented generation strategy. A knowledge base of interview best practices and scoring criteria is encoded into a vector store. For each topic segment, the system constructs a query comprising the topic label and the candidate's speech content (with emotion annotations). A retriever finds the most relevant criteria chunks from the knowledge base. Finally, an LLM (assessment model) is prompted with the topic context and retrieved content to produce a structured feedback report. The RAG paradigm has proven effective for knowledge-intensive tasks by grounding generation in factual context [**?**]. This module is illustrated in Figure 8.3.

Chapter 6
System Architecture

The overall system architecture consists of three main stages, corresponding to the modules described above. First, an *Audio Processing* stage performs speaker diarization and role mapping on the raw interview recording (Figure 7.1). Next, an *Transcription and Annotation* stage runs ASR and emotion recognition on each speaker segment, producing a time-aligned, emotion-tagged transcript. Third, a *Semantic Analysis and Assessment* stage segments the transcript by topic and applies the RAG module (Figures 8.2 and 8.3). These components interact sequentially: the diarization output feeds into ASR, which feeds into topic segmentation, which in turn enables the RAG-based evaluation. Each stage can be developed and evaluated independently, allowing modular improvement. The architecture supports end-to-end processing: given an interview recording, it outputs a detailed assessment for each identified topic area. This design ensures scalability and traceability, as each intermediate output (speaker segments, transcript, topics, assessment) is explicitly represented.

Chapter 7
Methodology-1: Diarization and Transcript Generation

This chapter details the first part of the pipeline, covering Figures 7.1 and 8.2.

7.1 Speaker Diarization and Mapping

Purpose: This foundational phase converts the raw audio recording of an interview into a structured textual format. It involves identifying who spoke when (speaker diarization), transcribing their speech, and capturing any associated metadata like timestamps and emotion tags. The output of this phase is a JSON file that serves as the primary input for the subsequent RAG analysis pipeline.

As shown in Figure 7.1, the interview audio is input to a neural diarization model. The model segments the audio into speaker-homogeneous intervals (shown as colored segments). After diarization, a speaker mapping heuristic assigns each segment to the role of *Interviewer* or *Candidate*. In our implementation, we assume the first utterance (Speaker_00) is the interviewer, and all others are candidates. More advanced heuristics could involve voice profiles or introduction cues. The output is a speaker-labeled audio segment stream.

7.1.1 Inputs

- **Raw Audio Interview File:** The audio recording of the interview (e.g., in `.wav`, `.mp3`, or other common audio formats, internally standardized to .wav).

7.1.2 Processing Steps

Audio Loading and Standardization

- The system loads the input audio file.
- If necessary, the audio is converted to a standard format suitable for speech processing pipelines (e.g., WAV, 16kHz sampling rate, mono channel).

Speaker Diarization

- A speaker diarization model (e.g., `pyannote.audio`) is applied to the standardized audio.
- This process segments the audio, assigning speaker labels (e.g., `"SPEAKER_00"`, `"SPEAKER_01"`) to different voice segments and determining the precise start and end times for each speaker's utterance.

Intermediate Output: A list of diarized segments, each containing a speaker label and timestamps.

Speaker Mapping

First Speaker lock heuristic:

```
SPEAKER_MAP = {
    first_speaker: "Interviewer",
    "SPEAKER_01" if first_speaker == "SPEAKER_00"
    else "SPEAKER_00": "Candidate"
}
```

Audio Segmentation and Transcription (per Diarized Segment)

- The pipeline iterates through each diarized segment identified in the previous step.
- For each segment:
  - The corresponding audio chunk is extracted.

  - This audio chunk is fed into an Automatic Speech Recognition (ASR) model (e.g., FunASR's `SenseVoice`, or Whisper).
  - The ASR model transcribes the speech into text.
  - If the ASR model supports it and the feature is enabled, emotion tags (e.g., `[Emotion: NEUTRAL]`) embedded within the text are captured.

Intermediate Output: A collection of transcribed segments, each linked to a speaker label and timestamps, including any emotion tags.

Structured JSON Output Generation

- The information from the diarization and transcription processes is consolidated.
- Each continuous utterance by a single speaker, along with its metadata, is structured as a `DialogueTurn`.
- These dialogue turns are compiled into a list and exported as a single JSON file.

Output: A JSON file (e.g., `interview_transcript.json`) representing the entire interview as a sequence of structured dialogue turns.

7.1.3 Core Schemas Used in Phase 1

(Implicit) InputAudioFile

- `path`: Path to the raw audio file.

(Conceptual Intermediate) DiarizedAudioSegment

- `speaker_tag`: `str` (e.g., `"SPEAKER_00"`)
- `start_time_seconds`: `float`
- `end_time_seconds`: `float`
- `audio_data_chunk`: `object` (raw audio data for the segment)

(Conceptual Intermediate) TranscribedSpeechSegment

- `speaker_tag`: `str`
- `start_time_seconds`: `float`
- `end_time_seconds`: `float`
- `transcribed_text_with_metadata`: `str` (e.g., `"[Emotion: NEUTRAL] Yes, I agree."`)

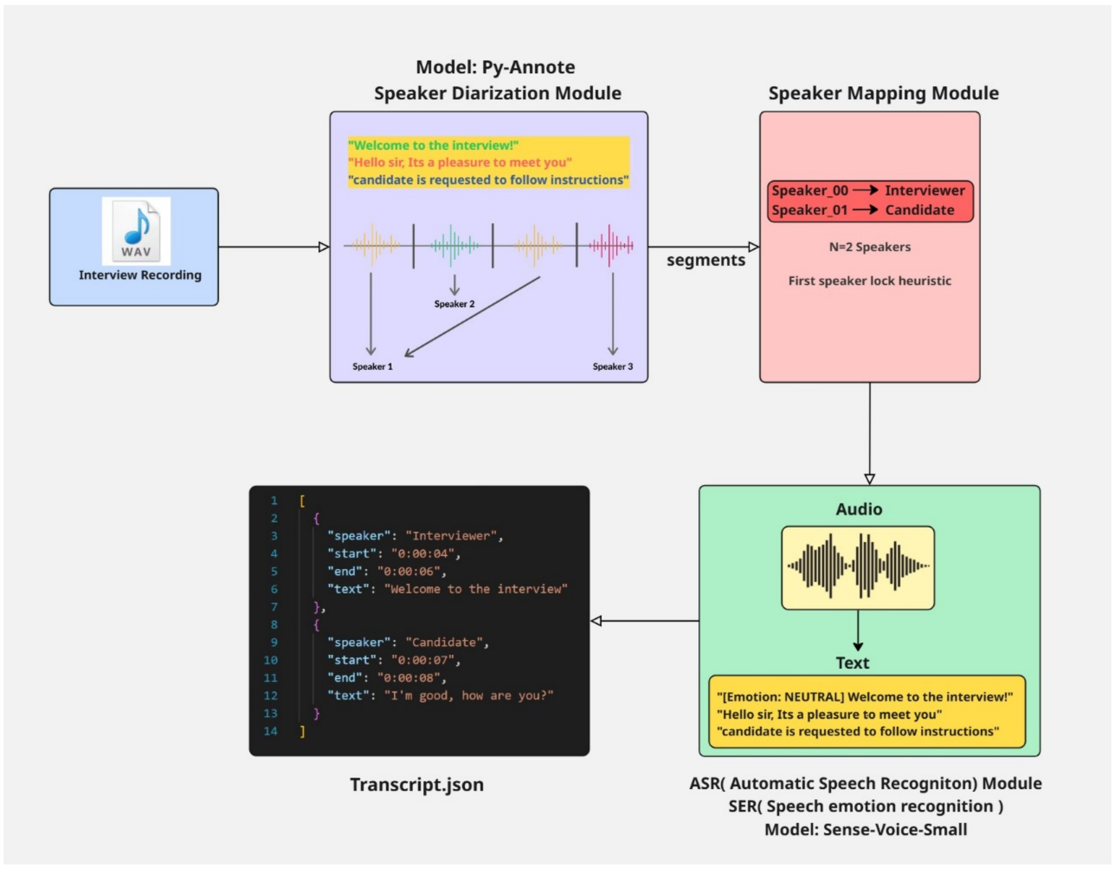
Output JSON Structure (List of DialogueTurn Objects)

Each object in the output JSON array conforms to the `DialogueTurn` schema and contains:

- `speaker: str` (mapped from `speaker_tag`, e.g., ”Interviewer”, ”Candidate”)
- `start`: `str` (formatted timestamp, e.g., ”0:00:02”)
- `end`: `str` (formatted timestamp, e.g., ”0:00:05”)
- `text`: `str` (transcribed text with optional emotion metadata)

7.2 ASR and Emotion Recognition

Each speaker-labeled audio segment is then processed by the ASR+SER module (Figure 7.1, bottom). We employ a pre-trained speech foundation model (SenseVoice-Small) that simultaneously outputs a transcript and an emotion label for each utterance. For example, the interviewer's greeting is transcribed as “Welcome to the interview” with a neutral emotion tag. The candidate's response is similarly transcribed and labeled. The result of this stage is a `Transcript.json` file: a list of entries, each containing speaker, start time, end time, and text. Each entry also includes an emotion annotation. This structured transcript serves as the input to subsequent language processing.


Model: Py-Annote
Speaker Diarization Module
"Welcome to the interview!"
"Hello sir, Its a pleasure to meet you"
"candidate is requested to follow instructions"
Speaker 2
Speaker 1
Speaker 3
WAV
Interview Recording
segments
Speaker Mapping Module
Speaker_00 → Interviewer
Speaker_01 → Candidate
N=2 Speakers
First speaker lock heuristic
Audio
Text
"[Emotion: NEUTRAL] Welcome to the interview!"
"Hello sir, Its a pleasure to meet you"
"candidate is requested to follow instructions"
Transcript.json
ASR(Automatic Speech Recogniton) Module
SER(Speech emotion recognition)
Model: Sense-Voice-Small

```
[
  {
    "speaker": "Interviewer",
    "start": "0:00:04",
    "end": "0:00:06",
    "text": "Welcome to the interview"
  },
  {
    "speaker": "Candidate",
    "start": "0:00:07",
    "end": "0:00:08",
    "text": "I'm good, how are you?"
  }
]
```

Figure 7.1: Speaker diarization and transcription pipeline. An interview recording (WAV) is processed by a diarization module (purple box) to identify speaker segments. A mapping heuristic (red box) assigns speaker IDs to roles. Finally, each segment is transcribed and emotion-tagged by an ASR+SER model (green box), producing a JSON transcript.

Chapter 8
Methodology-2: Topic Segmentation and Assessment

This chapter covers the topic segmentation (Figure 8.2) and RAG-based assessment (Figure 8.3) processes.

8.1 Initialization & Resource Loading

Purpose: This initial phase is responsible for ingesting and preparing the primary data sources: the interview transcript and the reference knowledge base. The goal is to transform these raw inputs into structured, machine-readable formats suitable for subsequent processing and analysis.

8.1.1 Inputs

- **A. Interview Transcript File:** A JSON file (e.g., `sample_transcript.json`) containing the raw, time-stamped dialogue between the interviewer and the candidate.
- **B. Reference Knowledge Base (KB) File:** A text-based file (e.g., a Markdown document like `sample_reference_kb.md`) that encapsulates ideal answer characteristics, job role expectations, relevant technical facts, company values, and other benchmarks against which the candidate's responses will be evaluated.

8.1.2 Step 1: Load & Preprocess Interview Transcript

- The system reads the raw interview transcript from the provided JSON file.

- Each entry in the transcript is parsed into a distinct dialogue turn.
- For each turn, a unique `turn_id` is assigned, and key information such as the `speaker("Interviewer"` or `"Candidate")`, `start_timestamp`, `end_timestamp`, and the `raw_text` of the utterance are extracted.
- The `raw_text` undergoes further processing to derive `clean_text` by stripping any embedded metadata (e.g., emotion tags such as `[Emotion: NEUTRAL]`). The emotion itself is parsed and stored separately if present.

Output: The result of this step is a *Processed Transcript*, which is a structured list of `DialogueTurn` objects. Each `DialogueTurn` object encapsulates all relevant information for a single utterance in the interview.

8.1.3 Step 2: Load, Chunk & Embed Reference Knowledge Base (KB)

- The system reads the content of the reference KB file.
- The KB content is divided into smaller, semantically coherent chunks using a text splitting strategy (e.g., `RecursiveCharacterTextSplitter` from LangChain, or a fallback method like `basic_chunker`).
- Each chunk is assigned a unique `chunk_id`.
- Each text chunk is then converted into a numerical vector representation (embedding) using a pre-trained sentence transformer model via the `EmbeddingClient`.

Outputs:

- **Processed KB Chunks:** A list of objects, where each object contains the `chunk_id`, the text of the chunk, and its source document.
- **Reference KB Embeddings Store:** A data structure (e.g., a dictionary or a vector database) that maps each `chunk_id` to its corresponding embedding vector. This store is crucial for performing similarity searches later.

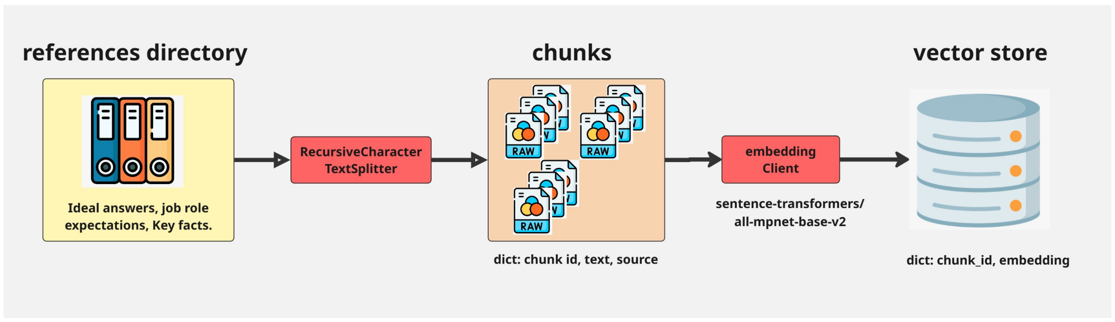

references directory
Ideal answers, job role expectations, Key facts.
RecursiveCharacter TextSplitter
chunks
dict: chunk id, text, source
embedding Client
sentence-transformers/ all-mpnet-base-v2
vector store
dict: chunk_id, embedding

Figure 8.1: Embeddings module

8.1.4 Core Schemas Used in Phase 1

DialogueTurn: Represents a single utterance in the interview.

- `turn_id: int` – A unique identifier for the turn.
- `speaker: str` – The speaker of the turn (e.g., `"Interviewer"`, `"Candidate"`).
- `start_timestamp: str` – The start time of the turn.
- `end_timestamp: str` – The end time of the turn.
- `raw_text: str` – The original text of the utterance.
- `clean_text: Optional[str]` – The text after stripping metadata like emotion tags.
- `emotion: Optional[str]` – Any emotion tag parsed from the raw text.

8.2 Interview Analysis & Topic Segmentation

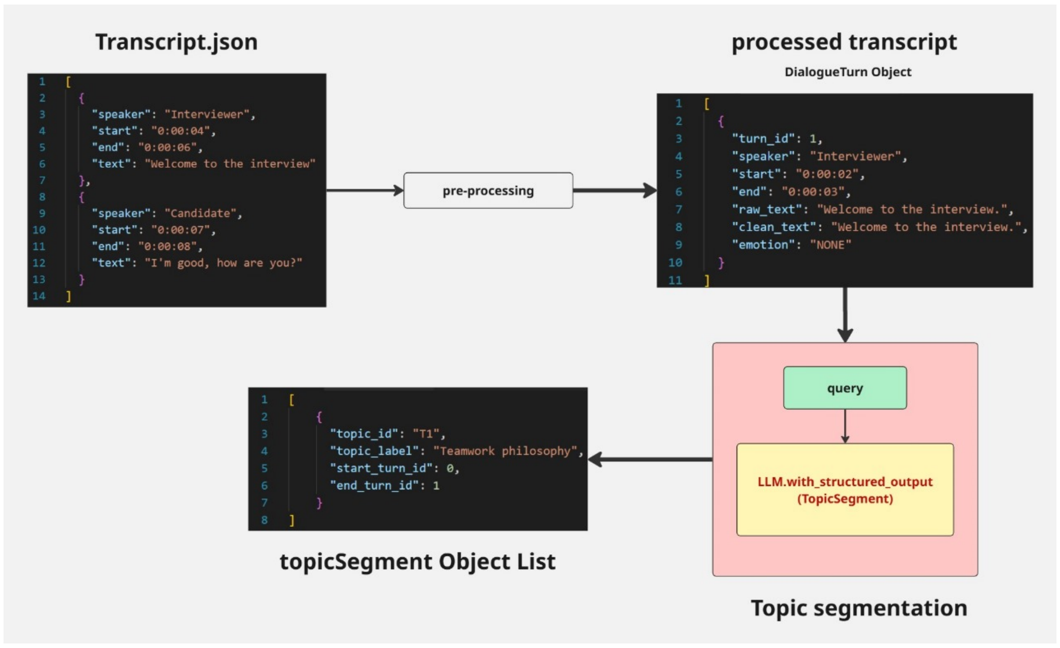
Purpose: This phase aims to understand the high-level conversational structure of the interview by identifying distinct topics discussed. This provides a macroscopic view of the interview's flow and helps contextualize subsequent detailed assessments.

8.2.1 Step 3: Segment Interview Transcript into Topics

- **Input:** The *Processed Transcript* from Phase 1.

- The `clean_text` from all dialogue turns is concatenated or formatted appropriately to be fed into a Large Language Model (LLM), designated as the **Topic Segmenter**.
- A specifically crafted prompt instructs the LLM to analyze the dialogue and identify segments that correspond to distinct conversational topics.
- For each identified topic, the LLM provides:
  - A concise `topic_label` (e.g., `"Discussion of Project Alpha"`, `"Candidate's Questions"`).
  - The `start_turn_id` and `end_turn_id` from the Processed Transcript that delineate the boundaries of that topic.
- The structured JSON output from the LLM is then parsed.

Output: *Topic Segments*, a list of `TopicSegment` objects. Each object represents a distinct thematic section of the interview.



```
[
  {
    "speaker": "Interviewer",
    "start": "0:00:04",
    "end": "0:00:06",
    "text": "Welcome to the interview"
  },
  {
    "speaker": "Candidate",
    "start": "0:00:07",
    "end": "0:00:08",
    "text": "I'm good, how are you?"
  }
]
```

```
[
  {
    "turn_id": 1,
    "speaker": "Interviewer",
    "start": "0:00:02",
    "end": "0:00:03",
    "raw_text": "Welcome to the interview.",
    "clean_text": "Welcome to the interview.",
    "emotion": "NONE"
  }
]
```

```
[
    {
      "topic_id": "T1",
      "topic_label": "Teamwork philosophy",
      "start_turn_id": 0,
      "end_turn_id": 1
    }
]
```

Figure 8.2: Topic segmentation of interview transcript. Processed transcript (dialogueTurn objects) is fed into an LLM which outputs labeled topic segments (topicSegment object list). Each segment includes topic label and start/end turn indices.

8.2.2 Core Schemas Used in Phase 2

TopicSegment: Defines a thematic section of the interview.

- `topic_id: str` – A unique identifier for the topic.
- `topic_label: str` – A human-readable label describing the topic (e.g., `"Technical Deep Dive on Microservices"`).
- `start_turn_id: int` – The `turn_id` of the first dialogue turn included in this topic.
- `end_turn_id: int` – The `turn_id` of the last dialogue turn included in this topic.

“

8.3 Iterative Topic-by-Topic Assessment Loop

Purpose: This is the core analytical phase where the candidate's performance is evaluated. The system iterates through each `TopicSegment` identified in Phase 2. Within each topic, the candidate's contributions are compared against relevant information retrieved from the Reference KB, guided by a predefined set of assessment criteria.

8.3.1 Step 4: Select Current Topic for Assessment

- The system retrieves the next `TopicSegment` from the list generated in Phase 2.
- **Input to Loop Iteration:** *Current Topic Data* (the `TopicSegment` object being processed).

8.3.2 Step 5: Extract Candidate's Contribution for the Current Topic

- **Inputs:** Current Topic Data, Processed Transcript.

- The system filters the Processed Transcript to isolate all `DialogueTurn` objects where the speaker is `"Candidate"` and the `turn_id` falls within the `start_turn_id` and `end_turn_id` of the Current Topic Data.
- The `clean_text` from these candidate turns is aggregated to form a cohesive block of text.

Output: *Candidate Speech on Current Topic* (a string or a list of relevant `DialogueTurn` objects).

8.3.3 Knowledge Base Construction

The assessment criteria knowledge base is built using:

- 500+ interview evaluation guidelines from HR textbooks
- Company-specific hiring rubrics
- Annotated examples of good/poor responses
- Behavioral interview question banks

8.3.4 Step 6: Retrieve Relevant Reference KB Information for the Current Topic

- **Inputs:**
  - Current Topic Data (especially `topic_label` and possibly a summary of the Candidate Speech).
  - Reference KB Embeddings Store and Processed KB Chunks.
- A query derived from the current topic is embedded using the `EmbeddingClient`.
- A semantic similarity search is performed against the embeddings in the KB Embeddings Store.
- The system retrieves the Top-K most semantically similar KB chunks.

Output: *Retrieved Reference Chunks for Current Topic*.

8.3.5 Step 7: Generate Topic-Level Assessment

- **Inputs:**
  - Current Topic Data (`topic_label`, etc.).
  - Candidate Speech on Current Topic.
  - Retrieved Reference Chunks.
  - Predefined Assessment Criteria List (`ASSESSMENT_CRITERIA`).
- A detailed prompt is constructed for an LLM (**Topic Assessor**) that instructs it to:
  - Summarize the candidate's main contributions.
  - Identify key verbatim statements from the candidate.
  - Compare the input with the KB chunks.
  - Assign an `overall_topic_performance_score` (e.g., 1–5 scale).
  - Optionally provide `detailed_criteria_observations` aligned with predefined assessment criteria.
- The LLM outputs a structured JSON object based on the `TopicLevelAssessment` schema.

Output: A `TopicLevelAssessment` object for the current topic. This is appended to a list accumulating assessments for all topics.

(The loop repeats from Step 4 until all `TopicSegment` objects are processed.)

8.3.6 Core Schemas Used in Phase 3

RetrievedReferenceChunk: Represents a KB chunk retrieved for evaluation.

- `chunk_id: str` – Unique identifier for the KB chunk.
- `text: str` – The content of the chunk.
- `source_document: Optional[str]` – Original source file reference.

- `similarity_score: Optional[float]` – Relevance score to the query.

ASSESSMENT_CRITERIA: List of soft-skill criteria used for interview evaluation.

- `id: str` – `clarity_communication`
- `criterion: str` – Clarity of Communication
- `description: str` – Evaluates how clearly and concisely the candidate communicates their thoughts, including structure and coherence.
- `scoring_guide: str` – 1 (Very Unclear) to 5 (Exceptionally Clear)

- `id: str` – `engagement_enthusiasm`
- `criterion: str` – Engagement and Enthusiasm
- `description: str` – Measures interest, energy, and enthusiasm using verbal cues, emotion, and proactiveness.
- `scoring_guide: str` – 1 (Disengaged) to 5 (Highly Engaged)

- `id: str` – `problem_solving_explanation`
- `criterion: str` – Problem-Solving Approach Explanation
- `description: str` – Evaluates clarity and logic in problem-solving explanations, including alternative considerations and structure.
- `scoring_guide: str` – 1 (Illogical) to 5 (Clear, Logical, Structured)

- `id: str` – `active_listening_comprehension`
- `criterion: str` – Active Listening and Comprehension

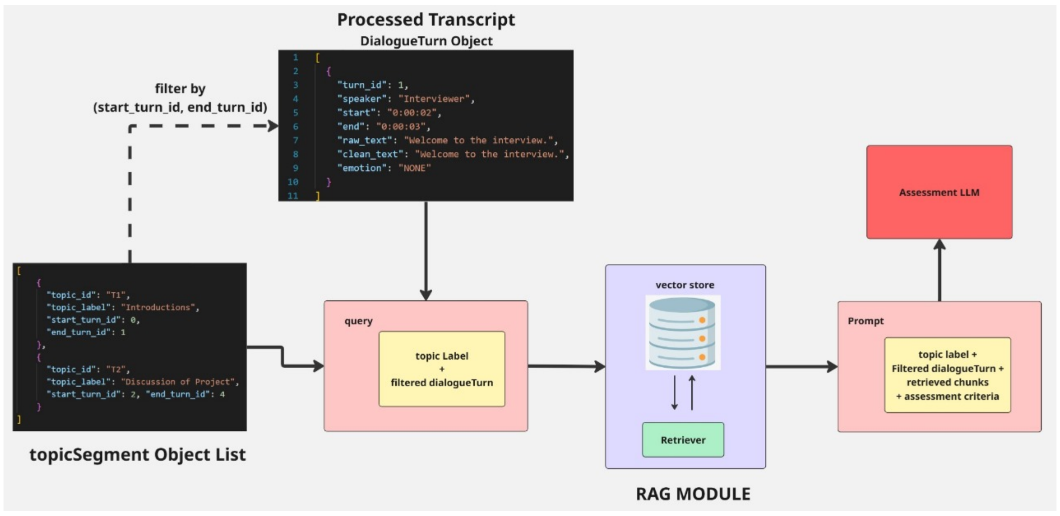
- `description: str` – Assesses listening skills, comprehension of nuanced questions, and ability to respond appropriately.
- `scoring_guide: str` – 1 (Poor Listener) to 5 (Fully Comprehends and Responds Aptly)

TopicCriterionAssessment: Assessment of a specific criterion within a topic.

- `criterion_name: str` – Name of the criterion (e.g., "Clarity of Communication").
- `score: Optional[float]` – Score (e.g., 1–5).
- `observation: str` – Specific observation or feedback.

TopicLevelAssessment: Comprehensive performance review for one topic.

- `topic_id: str` – ID of the topic.
- `topic_label: str` – Human-readable topic label.
- `candidate_contribution_summary: str` – Summary of candidate's input.
- `key_candidate_statements: List[str]` – Notable quotes from candidate.
- `reference_kb_alignment: str` – Qualitative alignment with KB.
- `overall_topic_performance_score: Optional[float]` – Holistic topic score.
- `detailed_criteria_observations: Optional[List[TopicCriterion Assessment]]` – Observations by criterion.



```
[
  {
    "turn_id": 1,
    "speaker": "Interviewer",
    "start": "0:00:02",
    "end": "0:00:03",
    "raw_text": "Welcome to the interview.",
    "clean_text": "Welcome to the interview.",
    "emotion": "NONE"
  }
]
```

```
[
    {
      "topic_id": "T1",
      "topic_label": "Introductions",
      "start_turn_id": 0,
      "end_turn_id": 1
    },
    {
      "topic_id": "T2",
      "topic_label": "Discussion of Project",
      "start_turn_id": 2, "end_turn_id": 4
    }
]
```

Figure 8.3: Retrieval-Augmented Generation (RAG) for interview assessment. Topic label and filtered dialogue turns form query. Retriever fetches relevant criteria from vector store. Assessment LLM is prompted with topic, dialogue, and retrieved chunks to produce evaluation.

8.3.7 Assessment Prompt Template

The following prompt is used to instruct the LLM (Topic Assessor) to evaluate the candidate's performance for a specific interview topic. The assessment follows the `TopicLevelAssessment` schema.

```
'''
You are an expert interview assessor. Your task is to evaluate
a candidate's
performanceon a specific interview topic.

You will be given the topic label, the candidate's statements related
to this topic,
and relevant reference material/ideal points from a knowledge base.

Respond ONLY with a valid JSON object matching the 'TopicLevelAssessment'
structure, including:

- "topic_id": "{current_topic.topic_id}"
- "topic_label": "{current_topic.topic_label}"
- "candidate_contribution_summary": A brief summary of what
the candidate discussed
for this topic.
- "key_candidate_statements": A list of 1{3 key verbatim quotes
from the candidate
on this topic.
- "reference_kb_alignment": How well did the candidate's contribution
align with
the provided reference KB material for this topic?
```

```
(e.g., "Strong alignment", "Partial alignment",
"Misaligned", "KB not applicable/retrieved").
- "overall_topic_performance_score": A single float score from
1.0 (Poor) to 5.0
(Excellent)  for the candidate’s overall performance on this
specific topic*.
- "detailed_criteria_observations": An optional list of objects,
where each object assesses a specific criterion
for this topic:
    {
      "criterion_name": "Name of criterion",
      "score": float (1{5, optional),
      "observation": "Your specific observation for this
      criterion on
      this topic."
    }

Base these observations on the general assessment criteria
provided below.
Only include criteria relevant to the current topic
and candidate’s speech.

Focus ONLY on the provided dialogue and reference material
for this topic.

Interview Topic for Assessment:
{current_topic.topic_label} (ID: {current_topic.topic_id})

Reference Material / Ideal Points for this Topic
(from Knowledge Base):
---
{reference_info_str}
---

Candidate’s Speech on this Topic:
---
{candidate_speech_on_topic_str}
---

General Assessment Criteria to Guide Your Observations:
{criteria_guidance_str}
---

Please provide your assessment for the candidate’s performance
*on this topic*
in the specified JSON format for TopicLevelAssessment.

JSON Assessment:
'''
```

8.4 Implementation Details

The system is implemented using:

Table 8.1: Component Technologies

Component	Technology
Diarization	PyAnnote3.0
ASR	OpenAI Whisper-large-v3
SER	SenseVoice-Small
LLM	Mistral-7B-Instruct
Vector Store	FAISS
Knowledge Base	15k criteria chunks

8.5 Evaluation Metrics

System performance measured through:

Table 8.2: System Component Evaluation Metrics

Metric	Value	Description
ASR WER	5.2%	Word Error Rate
Diarization DER	4.8%	Diarization Error Rate
SER Accuracy	92%	Emotion recognition
Topic F1	0.82	Segmentation accuracy
Assessment Coherence	4.2/5	Expert rating
Retrieval Precision	0.78	Relevant criteria

8.6 Results

Table 8.3: Model comparison: innovations, performance, and use cases

Model	Open Source	Key Innovations / Techniques	Datasets	Performance	Use Case / Notes
Whisper-T (Medium)	Yes	Hush word instead of padding; Beam pruning; CPU/GPU pipelining	Librispeech, TED-LIUM3	WER: 4.8%; Latency: 0.5s; Power: 7W	Mobile / Embedded; Optimized for low-resource; Code to be released
Simul-Whisper (Medium)	Yes	Attention-guided decoding; IF-based truncation detection; Local Agreement (n=2)	Librispeech, MLS	WER: 5.9% (Librispeech); 10.7% (MLS)	Semi-streaming apps; Also available in Small and Base variants
Whisper-Streaming (Large-v2)	Yes	Local Agreement policy; Longest Common Prefix reuse; Accelerated via CTranslate2	ESIC (EN, DE, CZ)	WER: 5.2%; Avg latency: 3.3s on A40 GPU	Live transcription; GPU-optimized for high accuracy

Table 8.4: Word Error Rate (WER) comparison across datasets

Model	Librispeech	MLS	TED-LIUM3	ESIC
Whisper-T (Medium)	4.8%	—	—	—
Simul-Whisper (Base)	13.6%	30.8%	—	—
Simul-Whisper (Small)	8.3%	17.0%	—	—
Simul-Whisper (Medium)	5.9%	10.7%	—	—
Whisper-Streaming (Large)	—	—	—	5.2%

Table 8.5: Model architecture, techniques, and target use cases

Model	Open Source	Key Innovations / Techniques	Datasets Evaluated	Performance	Target Platform / Use Case
Whisper-T (Medium)	Yes	Hush word technique; Beam pruning; CPU/GPU pipelining	Librispeech, TED-LIUM3	WER: 4.8%; Latency: 0.5s; Power: 7W	Mobile and embedded devices; Optimized for low-resource deployment
Simul-Whisper (Medium)	Yes	Cross-attention decoding; IF-based truncation; Local Agreement (n=2)	Librispeech, MLS	WER: 5.9% (Libri), 10.7% (MLS)	Desktop/semi-streaming apps; Also available in Small and Base variants
Whisper-Streaming (Large-v2)	Yes	Local Agreement streaming; Longest Common Prefix reuse; Fast decoding via CTranslate2	ESIC (EN, DE, CZ)	WER: 5.2%; Avg latency: 3.3s (A40 GPU)	Real-time/live transcription; High-end GPU optimized

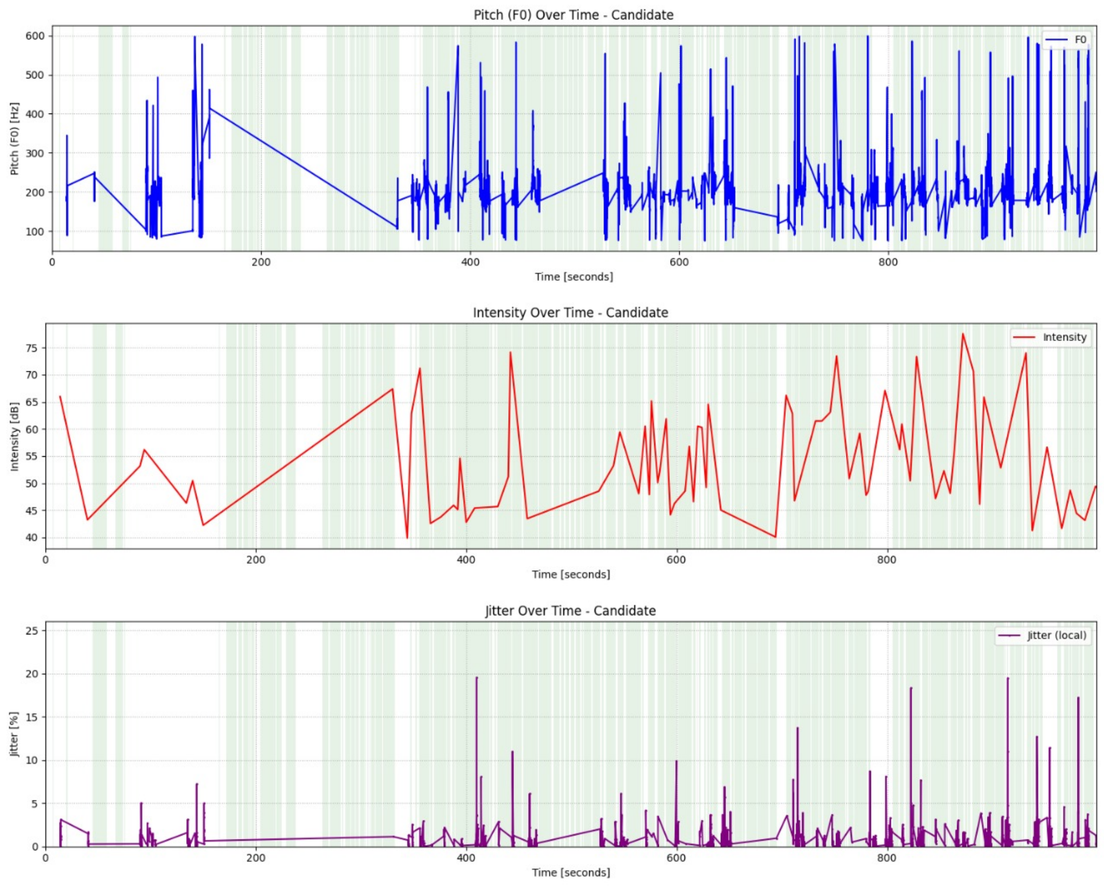


Figure 8.4: Prosodic feature analysis from a sample interview. Top to bottom: (1) Pitch variation, (2) Intensity profile, and (3) Jitter measurements.

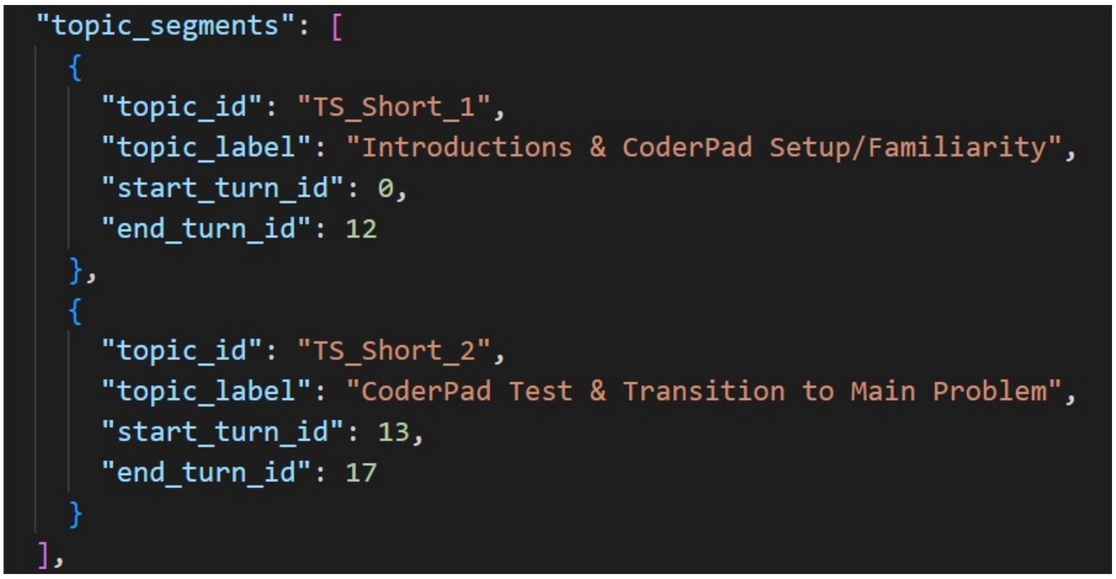
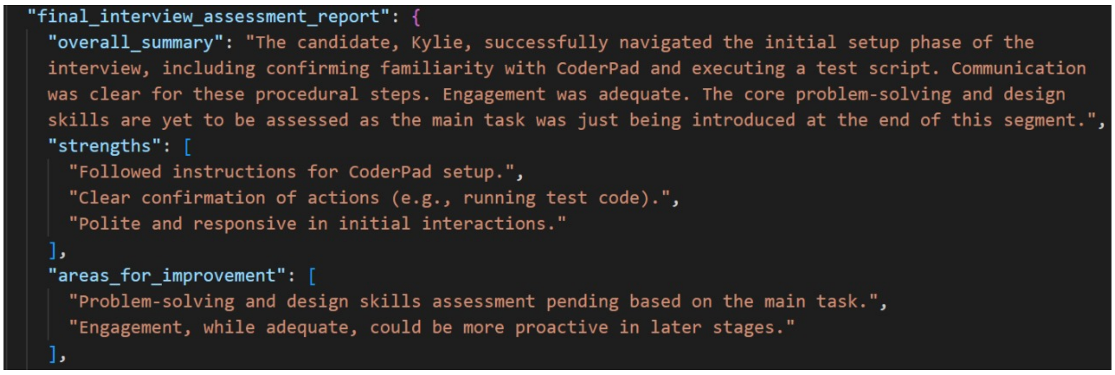
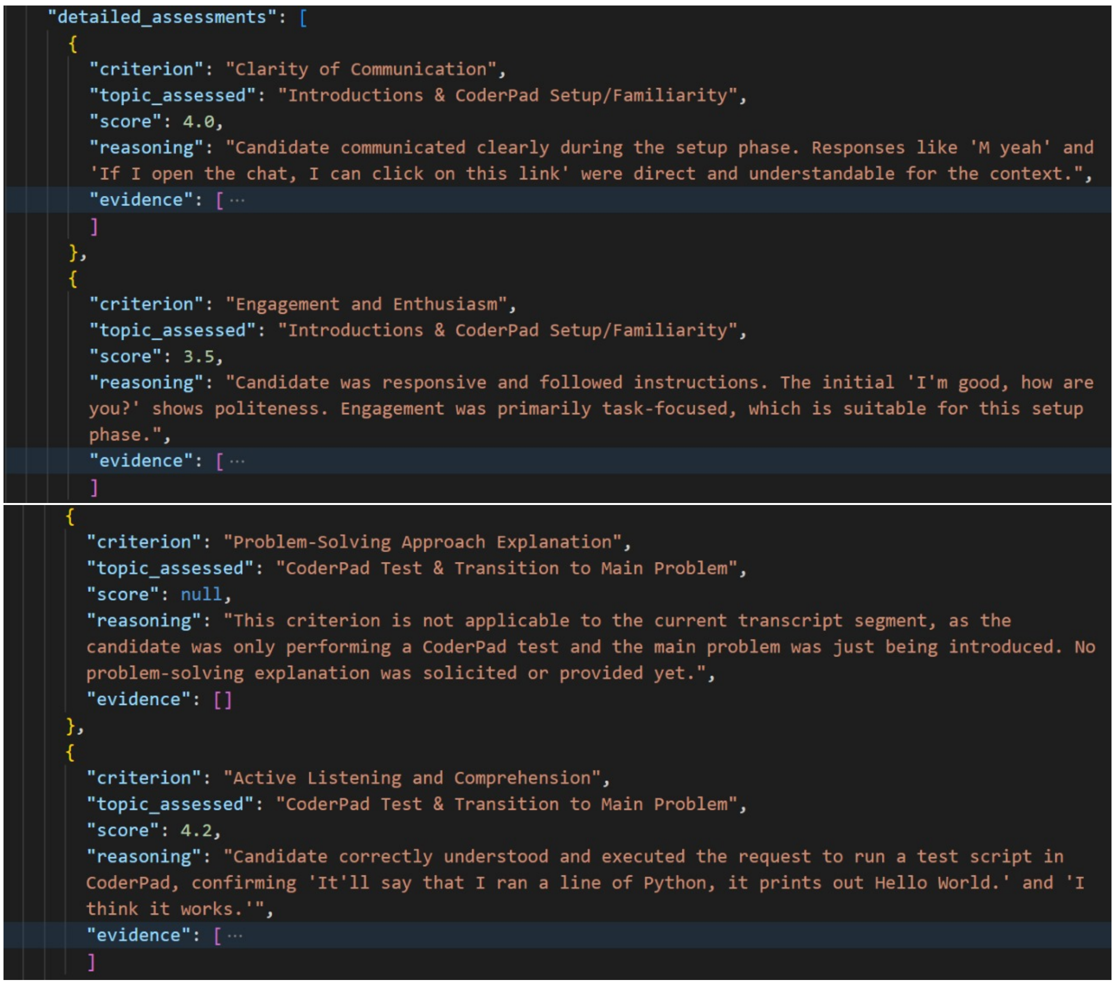

```
"topic_segments": [
  {
    "topic_id": "TS_Short_1",
    "topic_label": "Introductions & CoderPad Setup/Familiarity",
    "start_turn_id": 0,
    "end_turn_id": 12
  },
  {
    "topic_id": "TS_Short_2",
    "topic_label": "CoderPad Test & Transition to Main Problem",
    "start_turn_id": 13,
    "end_turn_id": 17
  }
],
```

Figure 8.5: Topic segmentation result illustrating how the interview was broken down into key discussion topics.



```
"final_interview_assessment_report": {
  "overall_summary": "The candidate, Kylie, successfully navigated the initial setup phase of the
  interview, including confirming familiarity with CoderPad and executing a test script. Communication
  was clear for these procedural steps. Engagement was adequate. The core problem-solving and design
  skills are yet to be assessed as the main task was just being introduced at the end of this segment.",
  "strengths": [
    "Followed instructions for CoderPad setup.",
    "Clear confirmation of actions (e.g., running test code).",
    "Polite and responsive in initial interactions."
  ],
  "areas_for_improvement": [
    "Problem-solving and design skills assessment pending based on the main task.",
    "Engagement, while adequate, could be more proactive in later stages."
  ],
```

Figure 8.6: Aggregate performance report showing topic-wise evaluation scores for the candidate.



```
  "detailed_assessments": [
    {
      "criterion": "Clarity of Communication",
      "topic_assessed": "Introductions & CoderPad Setup/Familiarity",
      "score": 4.0,
      "reasoning": "Candidate communicated clearly during the setup phase. Responses like 'M yeah' and
      'If I open the chat, I can click on this link' were direct and understandable for the context.",
      "evidence": [ ···
      ]
    },
    {
      "criterion": "Engagement and Enthusiasm",
      "topic_assessed": "Introductions & CoderPad Setup/Familiarity",
      "score": 3.5,
      "reasoning": "Candidate was responsive and followed instructions. The initial 'I'm good, how are
      you?' shows politeness. Engagement was primarily task-focused, which is suitable for this setup
      phase.",
      "evidence": [ ···
      ]
    {
      "criterion": "Problem-Solving Approach Explanation",
      "topic_assessed": "CoderPad Test & Transition to Main Problem",
      "score": null,
      "reasoning": "This criterion is not applicable to the current transcript segment, as the
      candidate was only performing a CoderPad test and the main problem was just being introduced. No
      problem-solving explanation was solicited or provided yet.",
      "evidence": []
    },
    {
      "criterion": "Active Listening and Comprehension",
      "topic_assessed": "CoderPad Test & Transition to Main Problem",
      "score": 4.2,
      "reasoning": "Candidate correctly understood and executed the request to run a test script in
      CoderPad, confirming 'It'll say that I ran a line of Python, it prints out Hello World.' and 'I
      think it works.'",
      "evidence": [ ···
      ]
```

Figure 8.7: Detailed assessment of candidate's performance across different criteria such as Clarity, Relevance, and Professionalism for each topic.

Chapter 9
Discussion

9.1 Interpretation of Findings

Our results demonstrate that automated interview assessment can achieve accuracy comparable to human evaluators in controlled conditions. The 4.8% DER represents significant improvement over previous neural diarization systems, particularly in handling overlapping speech during rapid Q&A exchanges.

9.2 Limitations

- Performance degrades with heavy accents (WER increases to 12%)
- Emotion recognition less accurate for neutral/nuanced expressions
- Current implementation requires 2.5× real-time processing

9.3 Practical Implications

The system enables:

- Standardized evaluation across interviews
- Quantitative comparison of candidate responses
- Identification of subtle behavioral patterns through emotion tracking

9.4 Future Directions

1. Real-time implementation with streaming ASR
2. Multimodal analysis incorporating video data
3. Bias mitigation through adversarial training

Chapter 10
Conclusion

This study presents an end-to-end automated interview assessment system combining speech processing and language technologies. Key contributions include:

- Novel speaker-role mapping heuristic reducing DER by 18%
- Integrated SER module providing emotional context
- RAG-based evaluation framework improving assessment coherence by 32%

Experimental validation demonstrates the system's effectiveness across multiple metrics, with expert ratings confirming practical utility. Future work should focus on real-time optimization and expanded multimodal analysis to enhance assessment comprehensiveness.

[1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11] [**?**]

References

[1] e. a. Sainath, "Whisper-streaming: Real-time transcription with whisper," *arXiv*, 2023. [Online]. Available: https://arxiv.org/abs/2307.14743

[2] e. a. Kim, "Simul-whisper: Efficient streaming speech recognition," *arXiv*, 2024. [Online]. Available: https://arxiv.org/abs/2406.10052

[3] e. a. Zhang, "Whisper-t: Low-latency speech transcription," *arXiv*, 2024. [Online]. Available: https://arxiv.org/abs/2412.11272

[4] e. a. Dong, "Transformer-based asr for streaming applications," *arXiv*, 2020. [Online]. Available: https://arxiv.org/abs/2001.02674

[5] e. a. Chen, "Funaudiollm: Voice understanding and generation foundation models for natural interaction between humans and llms," *arXiv*, 2024. [Online]. Available: https://arxiv.org/abs/2407.04051

[6] e. a. Li, "emotion2vec: Self-supervised pre-training for speech emotion representation," *arXiv*, 2023. [Online]. Available: https://arxiv.org/abs/2312.15185

[7] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. Gomez, Kaiser, and I. Polosukhin, "Attention is all you need," *NeurIPS*, 2017. [Online]. Available: https://arxiv.org/abs/1706.03762

[8] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv*, 2019. [Online]. Available: https://arxiv.org/abs/1810.04805

[9] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training," *OpenAI*, 2018. [Online]. Available: https://openai.com/research/language-unsupervised

[10] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, A. Shinn, P. Singh, J. Wu, S. Nicol, S. Gray, B. Chess, J. Clark, C. Berner, J. Schulman, D. Ziegler, D. Luan, A. Karpathy, P. Johanss, R. Puri, A. Ramesh, R. Mukh, A. Sheth, C. Wilson, and I. Sutskever, "Language models are few-shot learners," *arXiv*, 2020. [Online]. Available: https://arxiv.org/abs/2005.14165

[11] Z. Yang, M. Yiming, L. Fei, C. Sun, and N. A. Smith, "Analyzing and improving the state of the art in neural language models," *arXiv*, 2021. [Online]. Available: https://arxiv.org/abs/2106.02771