

Person 1 — Input & Audio Analysis: Step-by-step Plan

AI Interview System — Team Member: Aneesh

September 15, 2025

Contents

1 Executive summary	2
2 Objectives and scope	2
3 Deliverables (what to produce)	2
4 Acceptance criteria / Quality checks	3
5 Required tools and suggested libraries	3
6 Data and file organization (recommended)	3
7 Step-by-step plan	4
8 Debugging checklist and common failure modes	7
9 Week-by-week mapping (align to project timeline)	7
10 Communication integration with other team members	7
11 Appendix: Quick validation commands and checks	7

1. Executive summary

This document lists clear, actionable steps to implement the audio ingestion, ASR, transcript cleaning, VAD/segment alignment, and paralinguistic feature extraction parts of the project. Each step includes expected outputs, checks, and integration hooks so other team members can consume the results.

2. Objectives and scope

- Implement robust audio ingestion and normalization pipeline.
- Integrate ASR (Whisper / WhisperX or equivalent) producing timestamped transcripts.
- Clean and post-process transcripts (punctuation, filler removal, normalization).
- Produce utterance-level segmentation aligned to audio (VAD + silence-based chunking).
- Extract per-utterance paralinguistic features (pauses, speech rate, pitch stats, energy, MFCCs, stutter index).
- Define and produce standard output schemas (JSON/Parquet/CSV) and integration events for downstream modules.
- Provide evaluation metrics and baseline validation checks (WER computation, alignment checks, feature sanity tests).

3. Deliverables (what to produce)

1. **Code:** Modular, well-documented Python modules for ingestion, ASR wrapper, transcript post-processing, VAD/segmentation, and feature extraction.
2. **Artifacts:**
 - Timestamped transcripts (JSON) for sessions.
 - Segment-level feature files (Parquet/CSV/JSON) with standardized schema.
 - Example audio inputs + golden transcripts used for validation.
 - Unit tests covering core data transforms and a small evaluation script.
3. **Documentation:** README with setup instructions, expected input formats, API/event contracts for other teams, and acceptance criteria.
4. **Baseline metrics:** WER for a small validation set, coverage and alignment error statistics, and feature-distribution summaries.

4. Acceptance criteria / Quality checks

- ASR transcripts include timestamps per word/segment; WER on validation set $\leq 10\%$ (target, may be relaxed depending on data).
- For each response/utterance, there is a segment entry with: start, end, transcript, feature vector, and confidence score.
- No missing timestamps or negative durations; unit tests for basic schema checks pass.
- Feature extraction produces no NaNs for typical voiced segments and distributions look reasonable (e.g., mean pitch in plausible human ranges).
- Integration hooks (events or file locations) are stable and documented.

5. Required tools and suggested libraries

Languages / Runtime: Python 3.9+ (3.10 recommended).

Core libraries (suggested):

- ASR: `whisper`, `whisperx` (for alignment) or other cloud ASR APIs.
- Audio I/O: `ffmpeg` (system), `torchaudio`, `librosa`.
- VAD / Diarization (lightweight): `webrtcvad` (fast); for diarization use `pyannote.audio` (when needed by malpractice detector team).
- Feature extraction: `librosa` (pitch, energy, MFCC), `opensmile` (for richer paralinguistic features), optionally `pyAudioAnalysis`.
- Utilities: `numpy`, `pandas`, `pyarrow` (Parquet), `soundfile`.
- Evaluation: `jiwer` for WER, `pytest` for unit tests.

6. Data and file organization (recommended)

```
project/
  data/
    raw_audio/          # original uploads (wav, mp3)
    sampled_audio/      # standardized wav (16kHz, mono)
    transcripts/        # raw ASR outputs (JSON)
    segments/           # segment-level JSON/Parquet with features
    golden/             # small validation set with human transcripts
  src/
    ingest/
    asr_wrapper/
    postprocess/
    vad_segmentation/
    features/
    tests/
  docs/
  notebooks/
  outputs/
```

7. Step-by-step plan

Below are granular steps. For each step we list: *Goal*, *Tasks*, *Expected outputs*, and *Checks / Validation*.

Step i — Environment and baseline setup (Day 0–2)

Goal: Create reproducible environment and run a first-pass transcription on sample audio.

- Tasks:
 1. Create virtual environment and install dependencies (document versions).
 2. Ensure `ffmpeg` is installed on system PATH.
 3. Prepare 5–10 representative audio samples (different accents / SNRs / lengths).
 4. Run ASR (Whisper) on samples to produce timestamped transcript(s).
- Expected outputs: sample transcripts in `data/transcripts/`.
- Checks: transcripts have timestamps; basic WER computed against golden transcript(s) using `jiwer`.

Step ii — Standardize audio ingestion (Day 2–4)

Goal: Implement audio ingestion and normalization so downstream steps see consistent audio.

- Tasks:
 1. Build an ingest module that accepts common formats (wav/mp3/m4a).
 2. Convert all audio to 16 kHz, mono, 16-bit PCM using `ffmpeg`.
 3. Normalize amplitude (peak normalization) and trim leading/trailing silence (configurable thresholds).
 4. Add metadata capture (original filename, sample rate, duration, channels).
- Expected outputs: files in `data/sampled_audio/withmetadataJSON`.
- Checks: verify sample rates, durations, no corrupted files; log any failures.

Step iii — ASR wrapper and post-processing (Day 4–8)

Goal: Create a reusable ASR wrapper that supports alternate backends and produces cleaned, timestamped transcripts.

- Tasks:
 1. Wrap chosen ASR model (Whisper / WhisperX) with a consistent Python interface (functions: `transcribe_file`, `transcribe_segment`).
 2. Use WhisperX (or alignment tool) to get word-level timestamps where possible.
 3. Post-process transcripts to:
 - Restore punctuation (if model does not produce).

- Remove filler tokens ("uh", "um") configurable by a stoplist.
- Normalize whitespace and common contractions.
- Expected outputs: per-session JSON containing word-level timestamps and post-processed text.
- Checks: WER measured vs golden transcripts; retain raw ASR output separately for debugging.

Step iv — Utterance segmentation and alignment (Day 6–10)

Goal: Segment audio into candidate utterances (responses) that map cleanly to transcript spans.

- Tasks:
 1. Implement VAD-based segmentation (webrtcvad for speed; tune aggressiveness).
 2. Merge short voiced segments separated by very short pauses (configurable) to avoid over-segmentation.
 3. Align transcript timestamps to VAD segments: produce utterance objects {id, start, end, transcript, words[]}.
 4. Add a fallback: silence threshold chunking (if VAD fails on noisy data).
- Expected outputs: `segments/` JSON/Parquet with utterance-level start/end and transcripts.
- Checks: average words per segment reasonable (not zero), no huge gaps between transcript words and segment boundaries (≥ 1 s).

Step v — Paralinguistic feature extraction (Day 8–14)

Goal: Extract per-utterance audio features used by Emotion/Confidence and by Answer Evaluator.

- Tasks (per utterance):
 1. Pause metrics: pause before, pause after, number of internal pauses (≥ 50 ms) computed from silence splits.
 2. Speech rate: words per minute (using transcript word count and duration).
 3. Pitch: mean, median, standard deviation (use `librosa.pyin` or `praat` bindings).
 4. Energy: RMS mean/variance.
 5. MFCCs: first 13 coefficients aggregated (mean, std) across the utterance.
 6. Stutter / hesitation index: detect repeated short tokens ("I I I"), filler counts, long hesitations.
 7. Optional: jitter, shimmer, HNR (if using `praat` or `opensmile`).
- Expected outputs: feature vectors stored in a column `features` (nested JSON) or wide columns in Parquet.
- Checks: numeric ranges sanity, no NaNs for voiced segments, feature histograms produced for QA.

Step vi — Storage schema and integration hooks (Day 10–12)

Goal: Provide stable outputs other teams can consume and document event names or file locations.

- Suggested utterance JSON schema (one object per utterance):

```
{  
    "session_id": "...",  
    "utterance_id": "u_0001",  
    "speaker": "candidate",  
    "start": 12.345,  
    "end": 20.123,  
    "transcript": "I would use a hashmap because...",  
    "words": [{"text": "I", "start": 12.345, "end": 12.5}, ...],  
    "features": {"speech_rate_wpm": 64.2, "pitch_mean": 175.3, "mfcc_mean": [...], ...},  
    "asr_confidence": 0.88  
}
```

- Integration hooks (examples):

- Publish `transcript`, ready event with `session-level JSON`.
- Publish `segments`, ready event when `per-utterance features are available`.
- Place files in `outputs/` with stable naming convention `session-<id>-segments.parquet`.

- Checks: other team members can read and parse example JSON; add a small consumer script for testing.

Step vii — Metrics, evaluation and baseline (Day 12–15)

Goal: Provide baseline metrics that show pipeline quality.

- Compute WER using `jiwer` on golden set; log per-session and aggregate.
- Alignment error: mean absolute offset between ASR word timestamps and aligned utterance boundaries.
- Feature QA: produce summary statistics (mean, std, min, max) for key features across dataset.
- Produce a short report (not LLM, a plain summary) listing issues found and remediation steps.

Step viii — Tests, automation, and handoff (Day 14–18)

Goal: Make pipeline robust and easy to run; handoff to other team members.

- Implement unit tests for core transforms (ingest, resampling, segmentation mapping, feature extraction sanity).
- Create end-to-end script that consumes `data/raw_audio/and emits outputs/session-<id>-segments.parquet`
- Dockerfile or reproducible environment manifest (`requirements.txt / environment.yml`).
- Handoff docs: How to call the pipeline, expected outputs, and how to trigger reprocessing.

8. Debugging checklist and common failure modes

- Corrupted audio files: detect via sample length mismatch; skip and log.
- ASR timeouts / OOM: batch long files or use chunked transcription.
- Misaligned transcripts: fallback to chunk-level alignment (whisperx) or use forced-aligner.
- Too many short segments: tune VAD aggressiveness and merge rules.
- NaN features for very short/noisy segments: mark segments as *low quality and omit from feature-based models*.

9. Week-by-week mapping (align to project timeline)

Weeks	Tasks (Person 1)
1–2	Environment setup, collect sample audio, initial ASR runs, basic transcript output and WER baseline.
3	Implement ingest + normalization, ASR wrapper, basic VAD segmentation, initial feature extraction prototype.
4–5	Harden feature extraction (pitch, MFCCs, pause metrics), produce per-utterance features for training/evaluation, integrate with question retrieval team for sample exchanges.
6	Finalize schema, automation scripts, unit tests, baseline metrics and handoff documentation.

10. Communication integration with other team members

- Provide sample outputs (two full sessions) so Person 2 (Question Handler) and Person 3 (Evaluator) can validate assumptions.
- Share the JSON schema and a short consumer script so others can immediately read segments/feature files.
- Schedule one integration demo (30–45 minutes) after Week 3 to show transcripts + features in action.

11. Appendix: Quick validation commands and checks

- WER quick check: use `jiwer` to compute WER between ASR and golden transcripts.
- Sanity plots: pitch distribution (mean), speech rate histogram, pause-duration CDF.
- Schema validation: simple JSON Schema or a Pandas schema test to ensure required fields exist.